aws

即時串流使用者指南

Amazon IVS

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon IVS: 即時串流使用者指南

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon 的商標和商業外觀不得用於任何非 Amazon 的產品或服務，也不能以任何可能造成客戶混淆、任何貶低或使 Amazon 名譽受損的方式使用 Amazon 的商標和商業外觀。所有其他非 Amazon 擁有的商標均為其各自擁有者的財產，這些擁有者可能隸屬於 Amazon，或與 Amazon 有合作關係，或由 Amazon 贊助。

Table of Contents

什麼是 Amazon IVS 即時串流？

Amazon Interactive Video Service (IVS) 即時串流為您提供在應用程式中新增即時音訊和影片所需的一切。

強度：

- 即時延遲 – 針對延遲敏感的使用案例建置應用程式，協助您的觀眾保持連線並與 IVS 即時串流互動。提供即時串流，從主持人到觀眾之間的延遲時間不到 300 毫秒。
- 高並行性 – 透過 IVS 即時串流釋放大規模互動的潛力。容納多達 10,000 位觀眾的受眾，以及讓多達 12 位主持人進入虛擬舞台。
- 行動裝置優化 – IVS 即時串流已針對行動裝置使用案例進行優化，以迎合各種裝置和網路功能。藉由整合 Android 和 iOS 版 Amazon IVS 廣播 SDK，您的使用者能夠以主持人或觀眾的身分互動，在其行動裝置上享受高品質的即時串流。

使用案例：

- 訪客位置 – 建立可讓主持人「在舞台上」晉升訪客的應用程式，將觀眾轉變為主持人以進行即時互動。
- 對戰 (VS) 模式 – 透過並排競賽產生體驗，並讓觀眾即時觀看主持人競爭。
- 音訊室 – 邀請聽眾以訪客身分加入對話，並在您的音訊室內促進更深入的互動。
- 即時影片競拍 – 將競拍轉化為互動式影片事件，並透過即時延遲保持其興奮度和完整性。

除了此處的產品說明文件之外，請參閱 https://ivs.rocks/，此網站專供瀏覽已發佈的內容 (示範、程式碼範例、部落格文章)、估算成本，以及透過即時示範體驗 Amazon IVS。

全球解決方案、區域控制

串流和檢視是全球性的

您可以使用 Amazon IVS 以串流給世界各地的觀眾：

- 當您串流時，Amazon IVS 會在您附近的位置自動擷取影片。
- 觀眾可以在全球各地觀看您的直播。

另一種說法是「資料平面」是全球性的。資料平面是指串流/擷取和檢視。

控制是區域性的

雖然 Amazon IVS 資料平面是全球性的，但「控制平面」是區域性的。控制平面是指 Amazon IVS 主控台、API 和資源 (舞台)。

另一種說法是 Amazon IVS 是一種「區域性 AWS 服務」。也就是說，每個區域中的 Amazon IVS 資源與其他區域中的類似資源都是各自獨立的。例如，您在一個區域中建立的舞台與您在其他區域中建立的舞台無關。

當您使用資源 (例如，建立舞台) 時，您必須指定要建立舞台的區域。隨後，當您管理資源時，您必須從建立資源的相同區域執行這項操作。

如果您使用...	您可以指定區域，方法是透過...
Amazon IVS 主控台	使用導覽列右上角的 Select a Region (選擇區域) 下拉式清單。
Amazon IVS API	使用適當的服務端點。請參閱 Amazon IVS 即時串流 API 參考。 (如果您透過開發套件存取 API，請設定開發套件的 `region` 參數。請參閱在 AWS 上建置的工具。)
AWS CLI	任何一個： • 將 `--region <aws-region>` 附加到 CLI 命令。 • 將區域放入本機 AWS 組態檔案中。

請記住，無論舞台是在哪個區域建立，您都可以從任何地方串流至 Amazon IVS，觀眾也可以從任何地方觀看。

開始使用 IVS 即時串流

本文件將引導您完成將 Amazon IVS 即時串流整合到應用程式所涉及的步驟。

主題

簡介

必要條件

第一次使用即時串流之前，請先完成以下任務。如需說明，請參閱開始使用 IVS 低延遲串流。

- 建立 AWS 帳戶
- 設定根使用者和管理使用者。

其他參考

- IVS Web 廣播 SDK 參考
- IVS Android 廣播 SDK 參考
- IVS iOS 廣播 SDK 參考
- IVS 即時串流 API 參考

即時串流術語

術語	說明	
階段	參與者可即時交換影片的虛擬空間。	
主機	將本機影片傳送至階段的參與者。	
觀眾	接收主持人影片的參與者。	
參與者	以主持人或觀眾的身分連線至階段的使用者。	
參與者權杖	驗證參與者加入階段時的權杖。	
廣播 SDK	可讓參與者傳送和接收影片的用戶端程式庫。	

步驟概觀

1. the section called “設定 IAM 許可” – 建立 AWS Identity and Access Management (IAM) 政策，為使用者提供一組基本許可，並將該政策指派給使用者。
2. 建立階段 – 建立一個虛擬空間，參與者可以在這裡即時交換影片。
3. 分發參與者權杖 – 向參與者傳送記號，以便他們可以加入您的階段。
4. 整合 IVS 廣播 SDK – 將廣播 SDK 新增至應用程式，以便讓參與者傳送和接收影片：the section called “Web”、the section called “Android” 和 the section called “iOS”。
5. 發布和訂閱影片 – 將影片傳送至階段，並從其他主持人接收影片：the section called “Web”、the section called “Android” 和 the section called “iOS”。

設定 IAM 許可

接下來，您必須建立 AWS Identity and Access Management (IAM) 政策，為使用者提供一組基本許可 (例如，用於建立 Amazon IVS 階段並建立參與者權杖的許可)，並將該政策指派給使用者。您可以在建立新使用者時指派許可，也可以為現有的使用者新增許可。兩種程序如下所示。

如需詳細資訊 (例如，若要了解 IAM 使用者和政策、如何將政策附加到使用者、以及如何限制使用者可以使用 Amazon IVS 執行的動作)，請參閱：

- 《IAM 使用者指南》中的建立 IAM 使用者
- Amazon IVS 安全性 中有關 IAM 和「IVS 的受管政策」的資訊。
- Amazon IVS 安全性中的 IAM 資訊

您可以對 Amazon IVS 使用現有的 AWS 受管政策，也可以建立新政策來自訂要授予一組使用者、群組或角色的許可。兩種方法皆在下面有所描述。

對 IVS 許可使用現有的政策

在大多數情況下，您會想要對 Amazon IVS 使用 AWS 受管政策。IVS 安全性的 IVS 的受管政策章節對它們進行了完整描述。

- 使用 `IVSReadOnlyAccess` AWS 受管政策可讓應用程式開發人員存取所有 IVS Get 和 List API 端點 (同時用於低延遲和即時串流)。
- 使用 `IVSFullAccess` AWS 受管政策可讓應用程式開發人員存取所有 IVS API 端點 (同時用於低延遲和即時串流)。

選用：為 Amazon IVS 許可建立自訂政策

請遵循下列步驟：

1. 登入 AWS 管理主控台，然後前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在導覽窗格中，選擇政策，然後選擇建立政策。Specify permissions (指定許可) 視窗隨即開啟。
3. 在指定許可視窗中，選擇 JSON 標籤，將以下 IVS 政策複製並貼上至政策編輯器文字區域。(此政策不包括所有 Amazon IVS 動作。您可以視需要新增/刪除 (允許/拒絕) 端點存取許可。如需有關 IVS 端點的詳細資訊，請參閱 IVS 即時串流 API 參考。)

```
{
```

```
   "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. 仍在指定許可視窗中，選擇下一步 (捲動至視窗底部即可看到此選項)。Review and create (審查並建立) 視窗隨即開啟。

5. 在審查並建立視窗中，輸入政策名稱，並選擇性地新增描述。請記下政策名稱，因為您在建立使用者時要用到此資料 (如下)。選擇 Create policy (建立政策) (位於視窗底部)。
6. 您會返回 IAM 主控台視窗，在其中看到一個橫幅，確認您的新政策已建立。

建立新使用者並新增許可

IAM 使用者存取金鑰

IAM 存取金鑰由存取金鑰 ID 和私密存取金鑰組成。這些金鑰用於簽署您對 AWS 提出的程式設計請求。如果您沒有存取金鑰，可以使用 AWS 管理主控台來建立。根據最佳實務，請勿建立根使用者存取金鑰。

您只能在建立存取金鑰時，檢視或下載私密存取金鑰。稍後您便無法復原。不過，您可以隨時建立新的存取金鑰；您必須具有執行必要 IAM 動作的許可。

請務必安全地儲存存取金鑰。切勿與第三方分享 (即使查詢似乎來自 Amazon)。如需詳細資訊，請參閱《IAM 使用者指南》中的管理 IAM 使用者的存取金鑰。

程序

請遵循下列步驟：

1. 在導覽窗格中，選擇 Users (使用者)，然後選擇 Create user (建立使用者)。Specify user details (指定使用者詳細資訊) 視窗隨即開啟。
2. 在指定使用者詳細資訊視窗：
   a. 在 User details (使用者詳細資訊)，輸入要建立的新 User name (使用者名稱)。
   b. 勾選提供使用者對 AWS 管理主控台的存取權。
   c. 在 主控台密碼 下選取 自動產生的密碼。
   d. 選取使用者必須在下次登入時建立新密碼。
   e. 選擇下一步。Set permissions (設定許可) 視窗隨即開啟。
3. 在設定許可下，選取直接連接政策。Permissions policies (許可策略) 視窗隨即開啟。
4. 在搜尋方塊中，輸入 IVS 政策名稱 (AWS 受管政策或之前建立的自訂政策)。找到後，勾選方塊以選取政策。
5. 選擇 Next (下一步) (位於視窗底部)。Review and create (審查並建立) 視窗隨即開啟。
6. 在 Review and create (審查並建立) 視窗中，確認所有使用者詳細資訊都正確無誤，然後選擇 Create user (建立使用者) (位於視窗底部)。

7. Retrieve password (擷取密碼) 視窗隨即開啟，其中包含您的主控台登入詳細資訊。安全地儲存此資訊以供日後參考。當您完成時，請選擇 Return to users list (返回使用者清單)。

為現有的使用者新增許可

請遵循下列步驟：

1. 登入 AWS 管理主控台，然後前往 https://console.aws.amazon.com/iam/ 開啟 IAM 主控台。
2. 在導覽窗格中，選擇使用者，然後選擇要更新的現有使用者名稱。(按一下名稱來選擇名稱；請勿勾選選取方塊。)
3. 在 Summary (摘要) 頁面的 Permissions (許可) 索引標籤中，選擇 Add permissions (新增許可)。Add permissions (新增許可) 視窗隨即開啟。
4. 選取直接連接現有政策。Permissions policies (許可策略) 視窗隨即開啟。
5. 在搜尋方塊中，輸入 IVS 政策名稱 (AWS 受管政策或之前建立的自訂政策)。找到政策後，勾選方塊以選取政策。
6. 選擇 Next (下一步) (位於視窗底部)。Review (審查) 視窗隨即開啟。
7. 在檢閱視窗上，選取新增許可 (位於視窗底部)。
8. 在摘要頁面上，確認已新增 IVS 政策。

建立階段

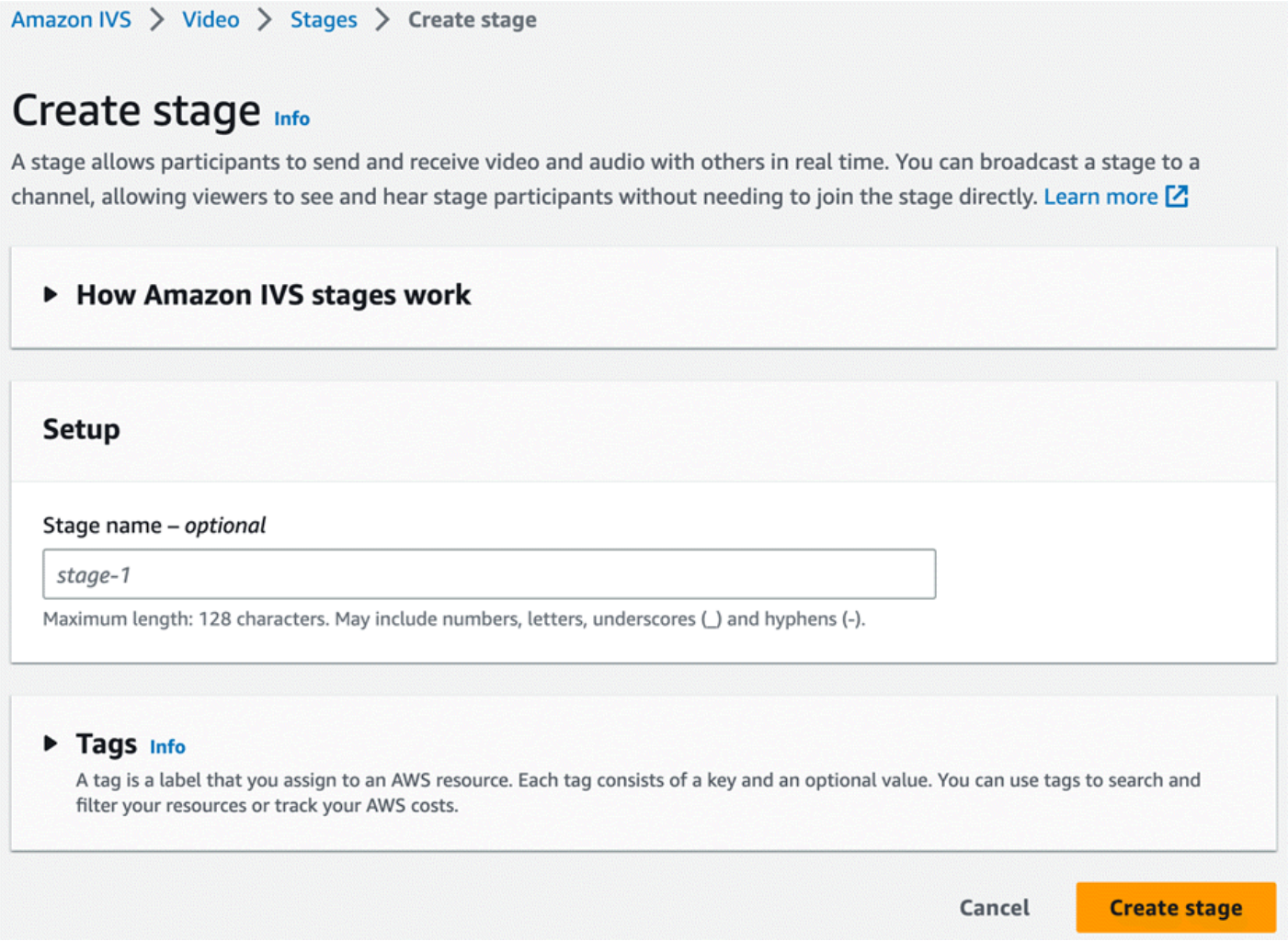
階段是參與者可即時交換影片的虛擬空間。它是即時串流 API 的基本資源。您可以使用主控台或 CreateStage 端點建立階段。

建議您儘可能針對每個邏輯工作階段建立一個新階段，並在完成後將其刪除，而不是保留舊階段供重複使用。如果未清除過時的資源 (舊階段，未重複使用)，則可能會更快地達到階段數目上限。

主控台說明

1. 開啟 Amazon IVS 主控台。

   (您也可以透過 AWS 管理主控台來存取 Amazon IVS 主控台。)
2. 在左側導覽窗格中，選取階段，然後選取建立階段。建立階段視窗出現。



3. 或者，輸入階段名稱。選取建立階段以建立階段。此時會顯示新階段的階段詳細資訊頁面。

CLI 說明

若要安裝 AWS CLI，請參閱安裝或更新最新版 AWS CLI。

您目前可以使用 CLI 來建立和管理資源。階段 API 位於 ivs-realtime 命名空間下方。例如，若要建立階段：

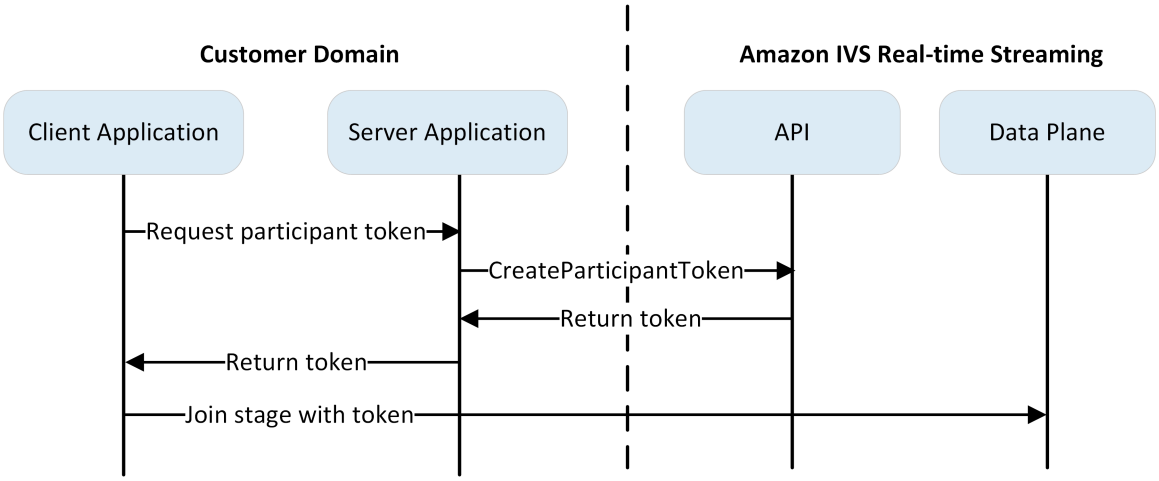
```
aws ivs-realtime create-stage --name "test-stage"
```

回應為：

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

```
      "name": "test-stage"
   }
}
```

分發參與者權杖



現在您有了一個階段，您需要建立權杖，並將其分發給參與者，以使他們能夠加入階段，並開始傳送和接收影片。

如上所示，客戶端應用程序詢問您的服務器應用程序的令牌，並且服務器應用程序 CreateParticipantToken 使用 AWS SDK 或 Sigv4 簽名請求調用。由於我們使用 AWS 憑證呼叫 API，因此應在安全的伺服器端應用程式中產生字符，而不是在用戶端應用程式中產生。

建立參與者權杖時，您可以選擇指定該權杖啟用的功能。預設值為 PUBLISH 和 SUBSCRIBE，這可讓參與者傳送和接收音訊和影片，但您可以簽發具有功能子集的權杖。例如，您可以簽發僅具有主持人適用 SUBSCRIBE 功能的權杖。在這種情況下，主持人可以看到正在傳送影片但不傳送自己影片的參與者。

您可以透過主控台或 CLI 建立參與者權杖，以進行測試和開發，但很可能您想要在生產環境中使用 AWS SDK 建立權杖。

您必須知道如何從伺服器將權杖分發給每位客戶 (例如透過 API 請求)。我們未提供此功能。依據本指南，您只需遵循下列步驟複製權杖並將其貼至用戶端程式碼。

重要：將權杖視為不透明；即，不根據權杖內容來建置功能。權杖的格式將來可能會改變。

主控台說明

1. 導覽至您在上一個步驟中建立的階段。

2. 選取建立參與者權杖。建立參與者權杖視窗即會出現。
3. 輸入要與權杖建立關聯的使用者 ID。這可以是任何 UTF-8 編碼文字。
4. 選取建立參與者權杖。
5. 複製字符。重要：請務必儲存權杖；IVS 不會存放該權杖，並且您稍後無法擷取。

CLI 說明

使用 AWS CLI 建立權杖，需要您先在機器上下載並設定 CLI。如需詳細資訊，請參閱《AWS 命令列介面使用者指南》。請注意，使用 AWS CLI 產生符記適合在測試時使用，但對於生產用途，建議您使用 AWS SDK 在伺服器端產生符記 (請參閱下述說明)。

1. 執行 `create-participant-token` 命令與階段 ARN。納入任何或所有這些功能：`"PUBLISH"`、`"SUBSCRIBE"`。

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

2. 這會傳回參與者權杖：

```
{
    "participantToken": {
        "capabilities": [
            "PUBLISH",
            "SUBSCRIBE"
        ],
        "expirationTime": "2023-06-03T07:04:31+00:00",
        "participantId": "tU06DT5jCJeb",
        "token":
 "eyJhbGciOiJLTVMiLCJ0eXAiOiJKV1QifQ.eyJleHAiOjE2NjE1NDE0MjAsImp0aSI6ImpGcFdtdmVFTm9sUyIsInJ
TaKjllW9Qac6c5xBrdAk"    }
}
```

3. 儲存此權杖。您將需要此項才能加入階段，並傳送和接收影片。

AWS SDK 說明

您可以使用 AWS SDK 來建立權杖。以下是 AWS SDK 的使用說明 JavaScript。

重要：此程式碼必須在伺服器端執行，再將其輸出傳遞給用戶端。

先決條件：若要使用下列程式碼範例，您必須安裝 aws-sd client-ivs-realtime k/ 套件。如需詳細資訊，請參閱的 AWS 開發套件入門 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);
```

整合 IVS 廣播 SDK

IVS 提供適用於 Web、Android 和 iOS 的廣播 SDK，並且您可以將其整合至應用程式。廣播 SDK 用於傳送和接收影片。在本節中，我們編寫了一個簡單的應用程式，讓兩個或更多參與者可以即時進行互動。以下步驟指導您創建名為的應用程序 BasicRealTime。完整的應用程序代碼已打開， CodePen 並且 GitHub：

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

Web

設定檔案

首先，建立資料夾和初始 HTML 和 JS 檔案來設定檔案：

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

您可以使用指令碼標籤或 NPM 來安裝廣播 SDK。為了簡單起見，範例使用指令碼標籤，但如果您稍後選擇使用 NPM，修改起來會很容易。

使用指令碼標籤

網路廣播 SDK 會以程式 JavaScript 庫的形式散發，可以在 https://web-broadcast.live-video.net/1.8.0/amazon-ivs-web-broadcast .js 擷取。

透過 `<script>` 標籤載入時，程式庫會在名為 `IVSBroadcastClient` 的視窗範圍中公開全域變數。

使用 NPM

安裝 NPM 套件：

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

您現在可以存取 IVS BroadcastClient 物件：

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

Android

建立 Android 專案

1. 在 Android Studio 中，建立新專案。
2. 選擇空白檢視活動。

   注意：在舊版 Android Studio 中，以檢視為基礎的活動被稱為空活動。如果您的 Android Studio 視窗顯示空活動，並確實不顯示空檢視活動，則請選取空活動。否則，請勿選取空活動，因為我們將使用 View API (而不是 Jetpack Compose)。
3. 為您的專案命名，然後選取完成。

安裝廣播 SDK

若要將 Amazon IVS Android 廣播程式庫新增到您的 Android 開發環境中，請將該程式庫新增到模組的 `build.gradle` 檔案，如下所示 (適用於 Amazon IVS 廣播 SDK 的最新版本)。在較新的專案中，`mavenCentral` 儲存庫可能已包含在您的 `settings.gradle` 檔案中，如果是這種情況，則您可以省略 `repositories` 區塊。在我們的範例中，也需要在 `android` 區塊中啟用資料繫結。

```
android {
    dataBinding.enabled true
```

```
}

repositories {
    mavenCentral()
}

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

或者，若要手動安裝 SDK，請從此位置下載最新版本：

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

iOS

建立 iOS 專案

1. 建立新的 Xcode 專案。
2. 若是平台，請選取 iOS。
3. 若是應用程式，請選取应用程式。
4. 輸入應用程式的產品名稱，然後選取下一步。
5. 選擇 (導覽至) 儲存專案的目錄，然後選取建立。

接著，您需要在 SDK 使用。我們建議您透過整合廣播 SDK CocoaPods。(或者，您可以手動將架構新增至您的專案中)。這兩種方法如下所述。

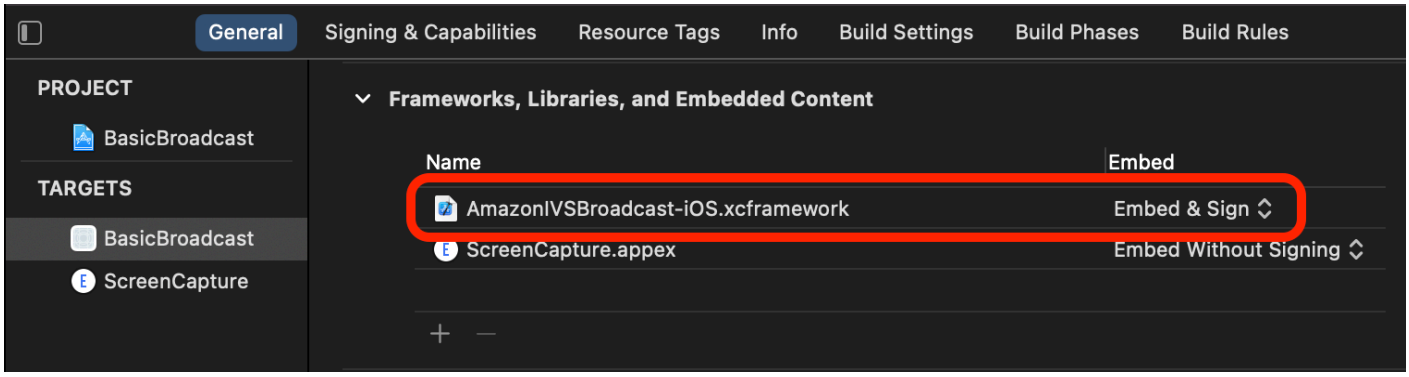
建議：安裝廣播 SDK (CocoaPods)

假設您的專案名稱是 `BasicRealTime`，在包含下列內容的專案資料夾中建立 `Podfile`，然後執行 `pod install`：

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

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

替代方法：手動安裝架構

1. 請從以下位置下載最新版本：https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip。
2. 解壓縮封存檔的內容。`AmazonIVSBroadcast.xcframework` 包含用於裝置和模擬器的 SDK。
3. 內嵌 `AmazonIVSBroadcast.xcframework`，方法是將其拖曳至您的應用程式目標的一般索引標籤的架構、程式庫和內嵌內容部分中：



設定許可

您需要更新您的專案 `Info.plist`，以新增 `NSCameraUsageDescription` 和 `NSMicrophoneUsageDescription` 的兩個項目。針對這些值，請提供解釋應用程式為何要求攝影機和麥克風存取權的面向使用者的說明。

Key	Type	Value
Information Property List	Dictionary	(3 items)
Application Scene Manifest	Dictionary	(2 items)
Privacy - Microphone Usage Description	String	We need access to your microphone to publish your audio feed
Privacy - Camera Usage Description	String	We need access to your camera to publish your video feed

發布和訂閱影片

有關網絡，安卓和 iOS 的詳細信息，請參閱以下詳細信息。

Web

建立 HTML 樣板

首先，建立 HTML 樣板並將程式庫匯入為指令碼標籤：

```
<!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>
```

接受權杖輸入並新增加入/離開按鈕

在這裡用輸入控制填寫內文。這些做為輸入權杖，然後設定加入和離開按鈕。通常，應用程式會透過您的應用程式 API 請求權杖，但在本範例中，您將複製權杖並將其貼至權杖輸入。

```
<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 />
```

新增媒體容器元素

這些元素將為本機和遠端參與者保留媒體。新增了一個指令碼標籤，來載入 app.js 中定義的應用程式邏輯。

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

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

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

這樣就完成了 HTML 頁面，並且您應在瀏覽器中載入 index.html 時看到此頁面：

建立 app.js

繼續來定義 app.js 檔案的內容。首先，透過 SDK 的全域匯入所有必要的屬性：

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

建立應用程式變數

建立變數來保留加入和離開按鈕 HTML 元素的參考，以及存放應用程式的狀態：

```
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;
```

建立 joinStage 1：定義函數並驗證輸入

該 joinStage 函數採用輸入權杖，建立與階段的連線，並開始發布從 getUserMedia 中擷取的影片和音訊。

首先，定義函數並驗證狀態和權杖輸入。我們將在接下來的幾個部分完善此功能。

```
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
};
```

建立 JoinSage 2：取得要發布的媒體

以下是將在階段上發布的媒體：

```
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]);
```

建立 JoinSage 3：定義階段策略並建立階段

此階段策略是 SDK 用於決定發布內容，以及要訂閱哪些參與者的決策邏輯核心。如需函數用途的詳細資訊，請參閱策略。

這項策略很簡單。加入階段後，發布剛剛擷取的串流，並訂閱每個遠端參與者的音訊和影片：

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

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

建立 JoinSage 4：處理階段事件並轉譯媒體

階段會發出許多事件。我們需要聆聽 `STAGE_PARTICIPANT_STREAMS_ADDED` 和 `STAGE_PARTICIPANT_LEFT` 來回轉譯和移除頁面中的媒體。更詳盡的事件集列於活動中。

請注意，我們在這裡建立了四個協助程式函數，來協助管理必要的 DOM 元素：`setupParticipant`、`teardownParticipant`、`createVideoEl` 和 `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;
}
```

建立 JoinSage 5：加入階段

最終加入階段以完成 joinStage 函數！

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

建立 leaveStage

定義 `leaveStage` 離開按鈕將調用的函數。

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

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

初始化輸入事件處理常式

將新增一個最後的函數至 `app.js` 檔案。當載入頁面並建立事件處理常式以加入和離開階段時，會立即調用此函數。

```
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
```

執行應用程式並提供權杖

此時，您可以在本地或與其他人分享網頁，打開頁面，然後輸入參與者權杖並加入階段。

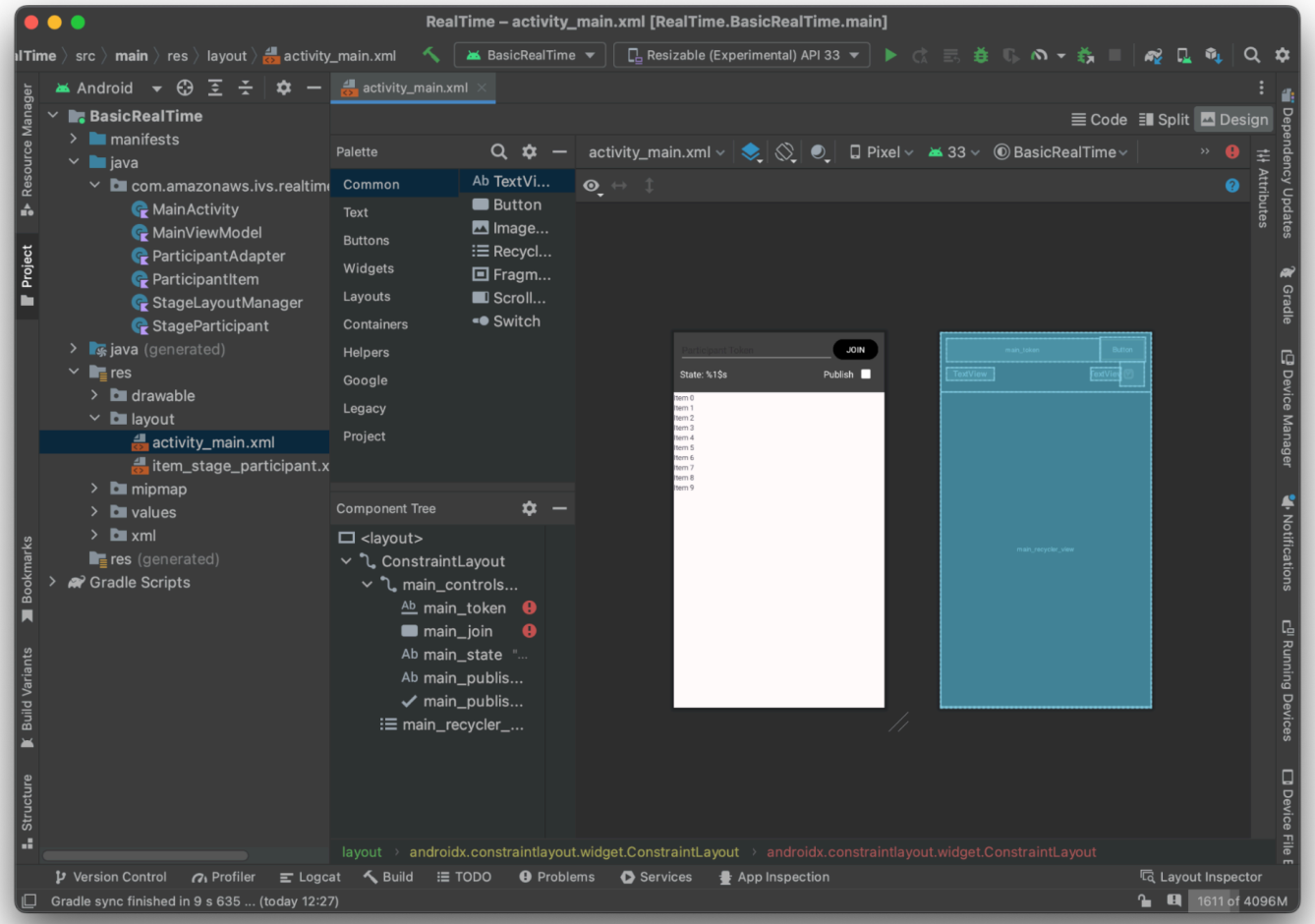
後續步驟？

有關 npm、React 等的更詳細範例，請參閱 IVS 廣播 SDK：網絡指南（即時串流指南）。

Android

建立檢視

首先使用自動建立的 `activity_main.xml` 檔案，來建立應用程式的簡單配置。配置包含一個可新增權杖的 `EditText`、一個加入 `Button`、一個可顯示階段狀態的 `TextView`，以及一個可切換發布的 `CheckBox`。



以下是檢視背後的 XML：

```
<?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>
```

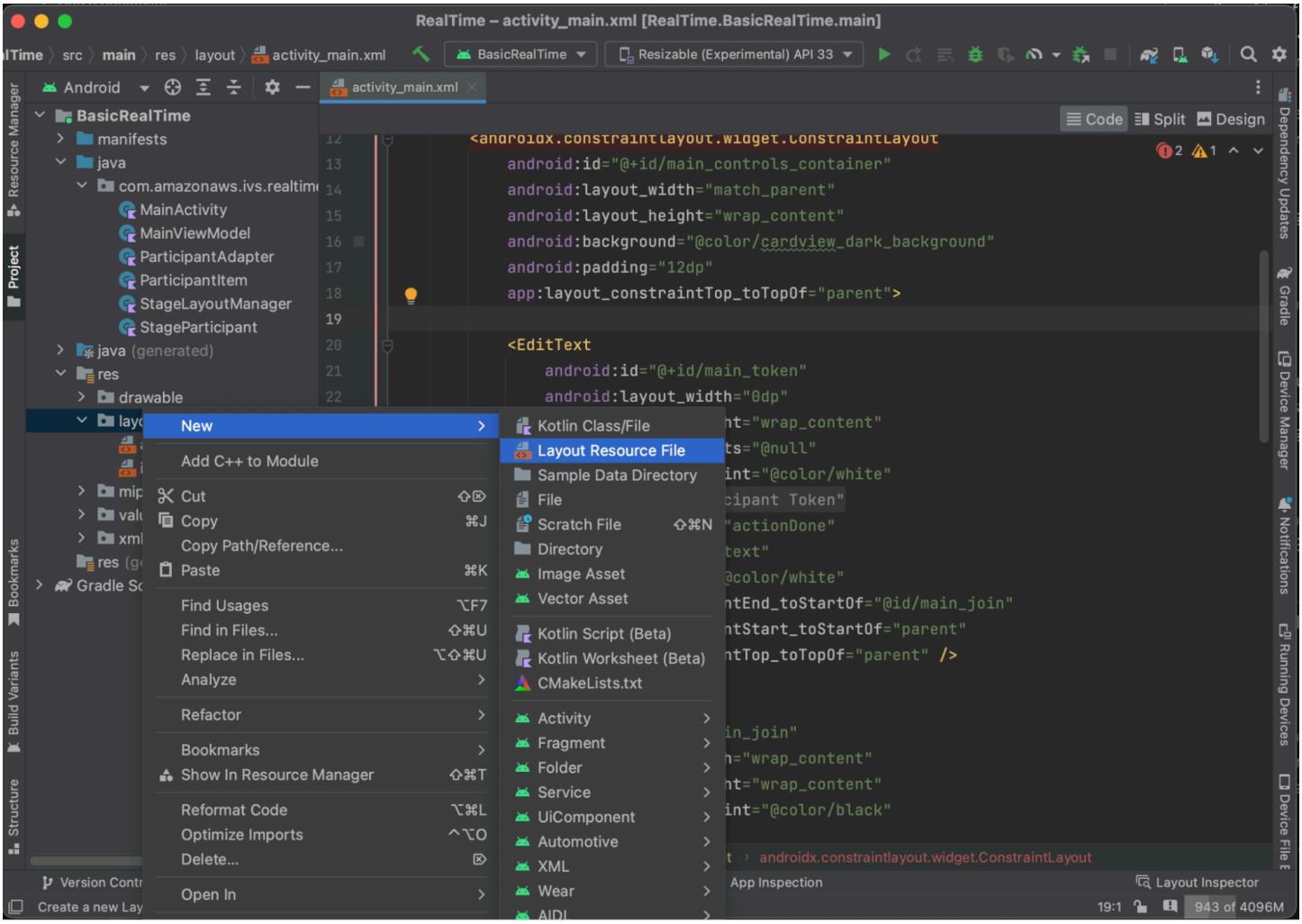
我們在這裡引用了幾個字串 ID，因此，現在將建立整個 `strings.xml` 檔案：

```
<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>
```

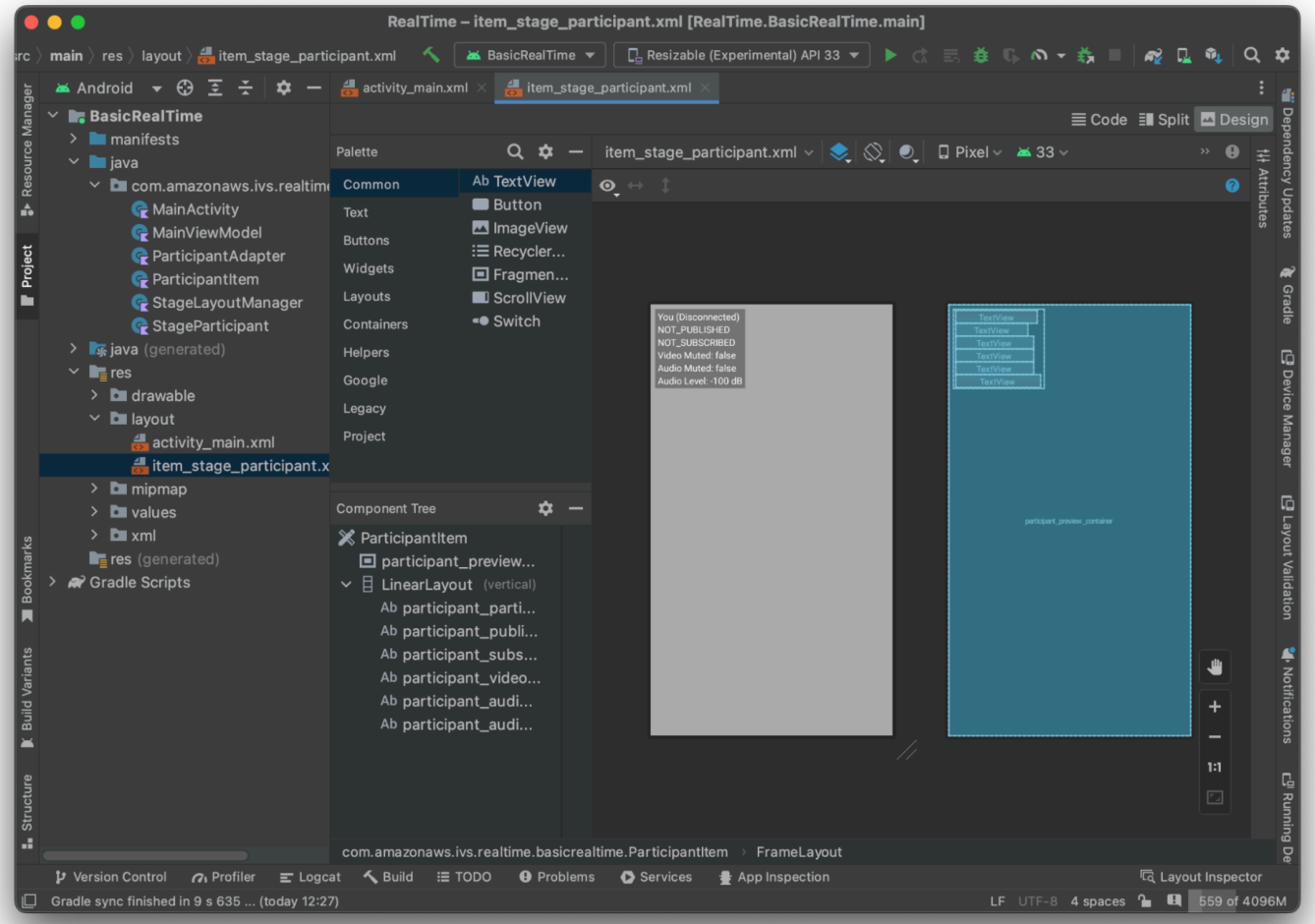
將 XML 中的這些檢視連結至 MainActivity.kt:

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

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

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

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

現在，將為 RecyclerView 建立項目檢視。若要執行此操作，在 res/layout 目錄上按一下滑鼠右鍵，然後選取新增 > 配置資源檔案。將此新檔案命名為 item_stage_participant.xml。



此項目的配置很簡單：包含一個用於轉譯參與者影片串流的檢視，以及一個用於顯示參與者資訊的標籤清單：



下面是 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>
```

此 XML 檔案擴充了尚未建立的類別 `ParticipantItem`。由於 XML 包括完整的命名空間，因此請務必將此 XML 檔案更新至您的命名空間。建立此類別並設定檢視，否則現在將其保留為空白。

建立一個新的 Kotlin 類別 `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)
    }
}
```

許可

若要使用攝影機和麥克風，您需要向使用者請求許可。我們對此遵循標準許可流程：

```
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
    }
}
```

應用程式狀態

我們的應用程序會跟踪本地的參與者，MainViewModel.kt並且該州將通信回MainActivity使用 Kotlin 的。StateFlow

建立一個新的 Kotlin 類別 MainViewModel:

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

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

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

}
```

將在 MainActivity.kt 中管理檢視模型：

```
import androidx.activity.viewModels

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

若要使用 AndroidViewModel 和這些 Kotlin ViewModel 延伸，您需要將以下內容新增至模組的 build.gradle 檔案:

```
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 適配器

我們將建立一個簡單的 `RecyclerView.Adapter` 子類別，以追蹤參與者並更新階段事件的 `RecyclerView`。首先，需要一個代表參與者的類別。建立一個新的 Kotlin 類別 `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)
            }
        }
}
```

我們將使用接下來會建立的 `ParticipantAdapter` 類別中的此類別。首先定義類別，並建立用於追蹤參與者的變數：

```
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>()
```

此外還必須定義 RecyclerView.ViewHolder，再實作其餘的覆寫：

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

使用此類別，可實作標準 RecyclerView.Adapter 覆寫：

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
    val item = LayoutInflater.from(parent.context)
        .inflate(R.layout.item_stage_participant, parent, false) as ParticipantItem
    return ViewHolder(item)
}

override fun getItemCount(): Int {
    return participants.size
}

override fun getItemId(position: Int): Long =
    participants[position]
        .stableID
        .hashCode()
        .toLong()

override fun onBindViewHolder(holder: ViewHolder, position: Int) {
    return holder.participantItem.bind(participants[position])
}

override fun onBindViewHolder(holder: ViewHolder, position: Int, payloads:
 MutableList<Any>) {
    val updates = payloads.filterIsInstance<StageParticipant>()
    if (updates.isNotEmpty()) {
        updates.forEach { holder.participantItem.bind(it) // implemented later }
    } else {
        super.onBindViewHolder(holder, position, payloads)
    }
}
```

最後，我們新增一些新的方法，當對參與者做出變更時，將會從 MainViewModel 中呼叫這些方法。這些方法是轉接器上的標準 CRUD 作業。

```
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])
    }
}
```

回到 MainViewModel，我們需要建立並保留此轉接器的參考：

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

階段狀態

還需要追蹤 MainViewModel 內的一些階段狀態。現在，我們來定義這些屬性：

```
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>()
```

若要在加入階段之前查看您自己的預覽，我們會立即建立本機參與者：

```
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)
}
```

我們希望確保在清除 ViewModel 時也清除這些資源。我們會馬上覆寫 onCleared()，因此不要忘了清除這些資源。

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

現在，一旦授予許可，我們會填充本機 streams，實作之前呼叫的 permissionsGranted 方法：

```
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)
    }
}
```

實作階段 SDK

以下是三個以即時功能為基礎的核心概念：階段、策略和轉譯器。設計目標是盡可能減少打造工作產品所需的用戶端邏輯數量。

Stage.Strategy

`Stage.Strategy` 實作很簡單：

```
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
}
```

總而言之，根據內部的 `publishEnabled` 狀態來發布，如果要發布，我們會發布之前收集的串流。最後，針對此範例，始終會訂閱其他參與者，同時接收其音訊和影片。

StageRenderer

StageRenderer 實作也非常簡單，但要考慮包含相當多程式碼的函數數量。當 SDK 通知我們有關參與者的變更時，此轉譯器中的一般方法是更新 ParticipantAdapter。在某些情況下，對於本機參與者的處理方式有所不同，因為我們已決定自行管理他們，以便在他們加入之前查看其攝影機預覽。

```
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) {}
}
```

實現自定義 RecyclerView LayoutManager

配置不同數量的參與者可能很複雜。您希望他們佔用整個父檢視的框架，但不想單獨處理每個參與者配置。為了簡化這一點，我們將逐步實作 RecyclerView.LayoutManager。

建立另一個新類別 StageLayoutManager，這應當會延伸 GridLayoutManager。設計此類別的目的是，根據以流程為基礎的資料列/資料欄配置中的參與者人數，來計算每個參與者的配置。每個資料列的高度與其他資料列相同，但每個資料列的寬度可能有所差異。請參閱 layouts 變數上方的程式碼註解，描述了如何自訂此行為。

```
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
}
```

回到 MainActivity.kt，我們需要為 RecyclerView 設定轉接器和配置管理器：

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

掛接 UI 動作

即將結束；我們只需掛接幾個 UI 動作。

首先，讓 MainActivity 觀測 MainViewModel 中的 StateFlow 變更：

```
// 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)
```

```
        }
    }
}
```

接著，將偵聽器新增至加入按鈕和發布核取方塊：

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

上述兩個呼叫功能均位於 `MainViewModel`，現在將在此實作：

```
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
}
```

轉譯參與者

最後，我們需要將從 SDK 接收的資料，轉譯到之前建立的參與者項目上。我們已經完成了 RecyclerView 邏輯，因此只需實作 ParticipantItem 中的 bind API。

從新增空函數開始，然後逐步實作：

```
fun bind(participant: StageParticipant) {

}
```

首先，將處理簡易狀態、參與者 ID、發布狀態和訂閱狀態。對於這些，只需直接更新 TextViews：

```
val participantId = if (participant.isLocal) {
    "You (${participant.participantId ?: "Disconnected"})"
} else {
    participant.participantId
}
textViewParticipantId.text = participantId
textViewPublish.text = participant.publishState.name
textViewSubscribe.text = participant.subscribeState.name
```

接著，將更新音訊和影片靜音狀態。為取得靜音狀態，需要從串流陣列中找到 ImageDevice 和 AudioDevice。為優化效能，我們會記住上次連接的裝置 ID。

```
// This belongs outside the `bind` API.
private var imageDeviceUrn: String? = null
private var audioDeviceUrn: String? = null

// This belongs inside the `bind` API.
val newImageStream = participant
    .streams
    .firstOrNull { it.device is ImageDevice }
textViewVideoMuted.text = if (newImageStream != null) {
    if (newImageStream.muted) "Video muted" else "Video not muted"
} else {
    "No video stream"
}

val newAudioStream = participant
    .streams
    .firstOrNull { it.device is AudioDevice }
```

```
textViewAudioMuted.text = if (newAudioStream != null) {
    if (newAudioStream.muted) "Audio muted" else "Audio not muted"
} else {
    "No audio stream"
}
```

最後，要轉譯 imageDevice 的預覽：

```
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
```

然後顯示 audioDevice 中的音訊統計資料：

```
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

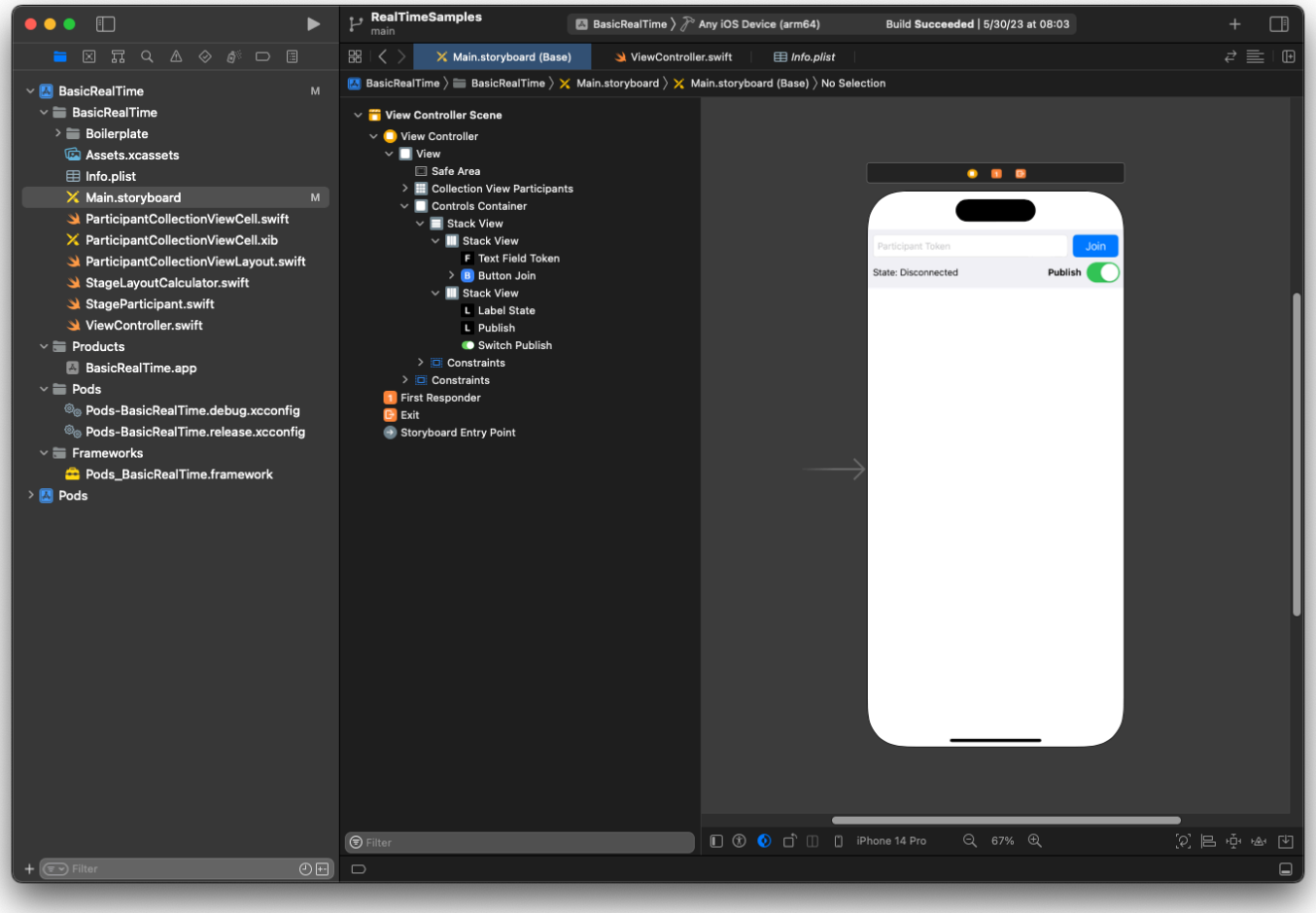
建立檢視

首先，使用自動建立用於匯入 AmazonIVSBroadcast 的 ViewController.swift 檔案，然後新增一些要連結的 @IBOutlets：

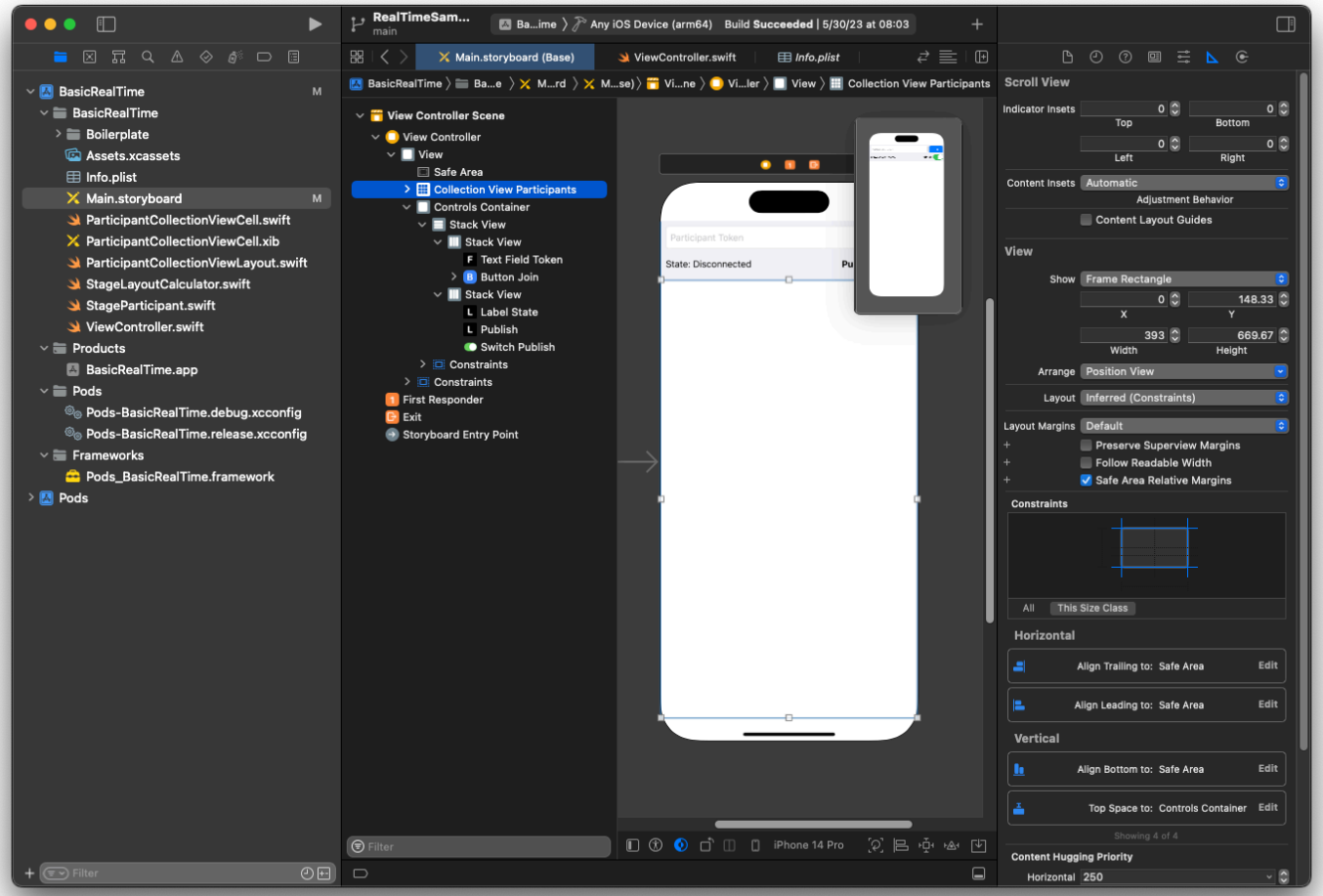
```
import AmazonIVSBroadcast
```

```
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!
```

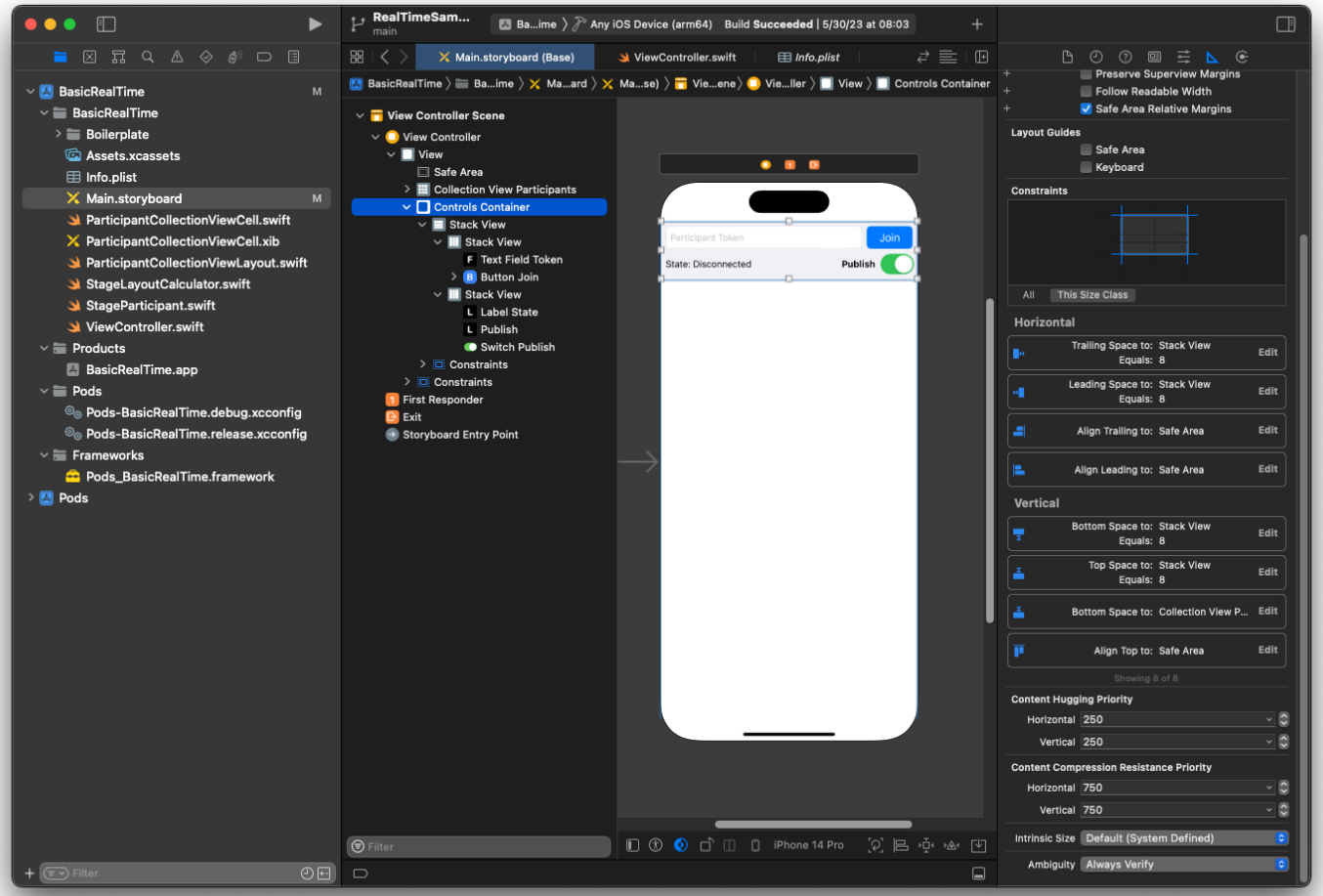
現在，建立這些檢視，並在 Main.storyboard 中將其連結。以下是將使用的檢視結構：



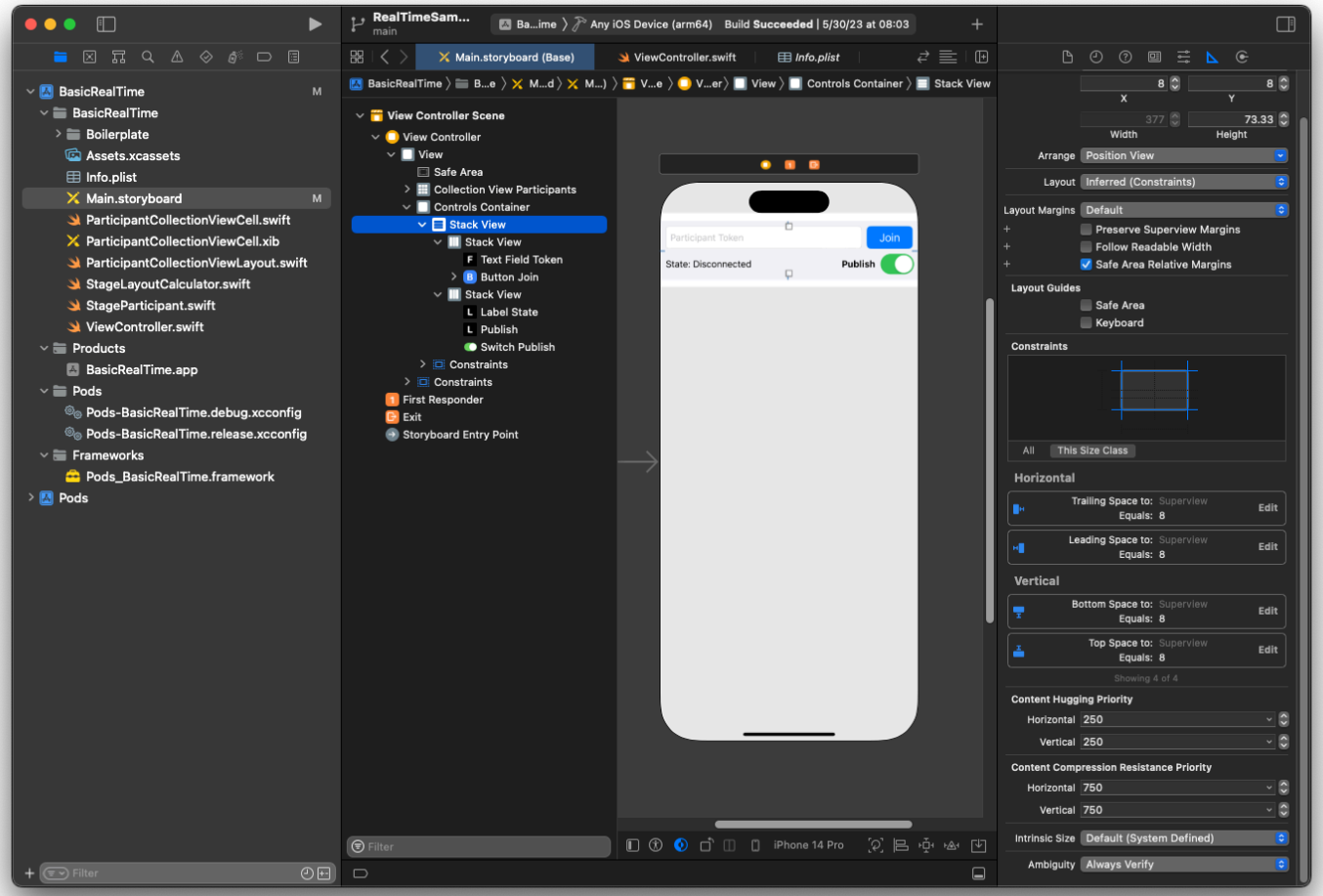
對於 AutoLayout 配置，我們需要自定義三個視圖。第一個檢視是集合檢視參與者 (UICollectionView)。將前導、尾隨和底部繫結至安全區域。同時將頂部繫結至控制容器。



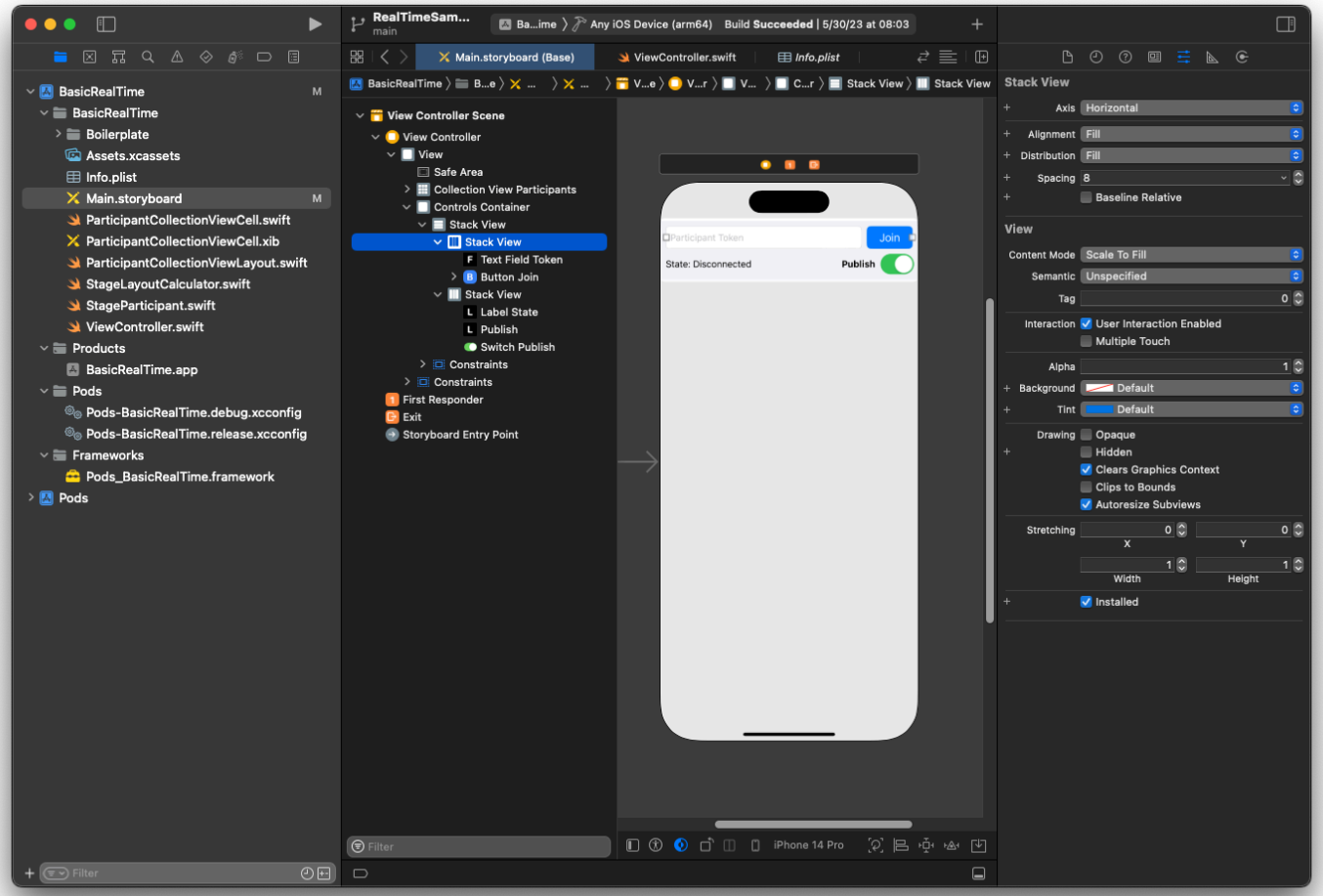
第二個檢視是控制容器。將前導、尾隨和頂部繫結至安全區域。



第三個和最後一個檢視是垂直堆疊檢視。將頂部、前導、尾隨和底部繫結至父檢視。針對樣式，將間距設定為 8，而不是 0。



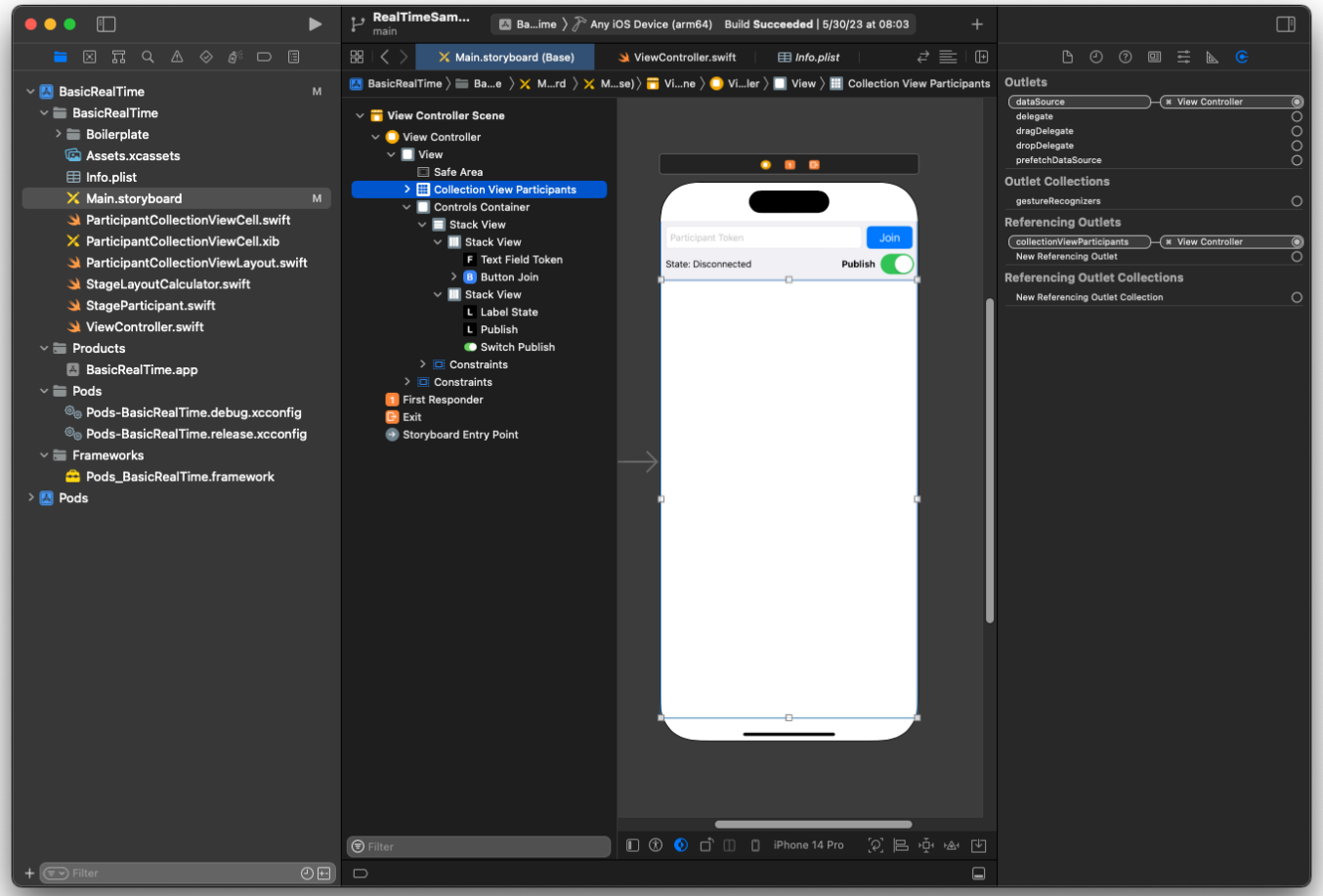
用戶界面StackViews將處理剩餘視圖的佈局。對於所有三個 UI StackViews，使用填充作為對齊和分佈。



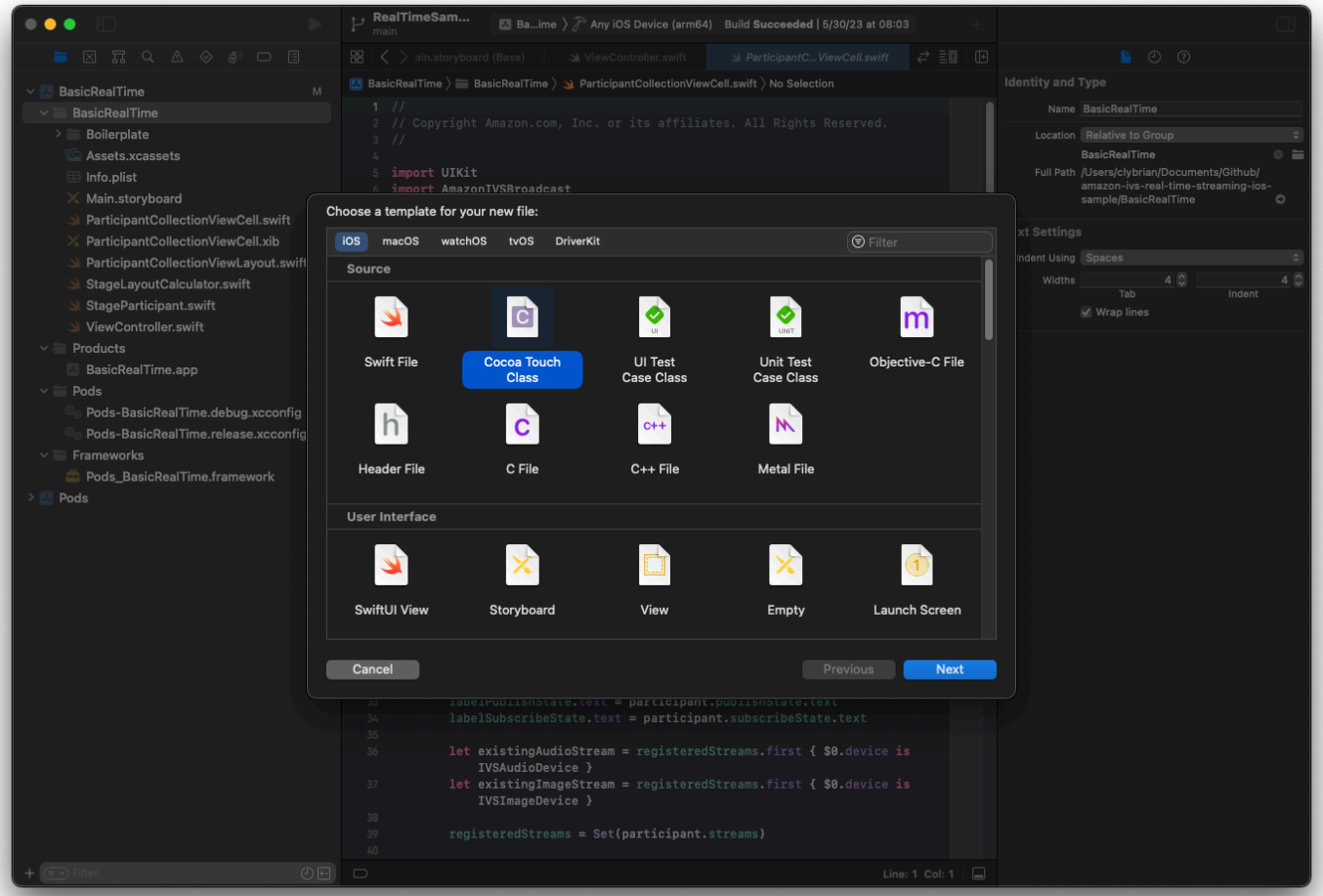
最後，將這些檢視連結至 ViewController。從上方映射下列檢視：

- 文字欄位加入繫結至 textFieldToken。
- 按鈕加入繫結至 buttonJoin。
- 標籤狀態繫結至 labelState。
- 切換發布繫結至 switchPublish。
- 集合檢視參與者繫結至 collectionViewParticipants。

同時還會將集合檢視參與者項目的 dataSource 設定為擁有的 ViewController:



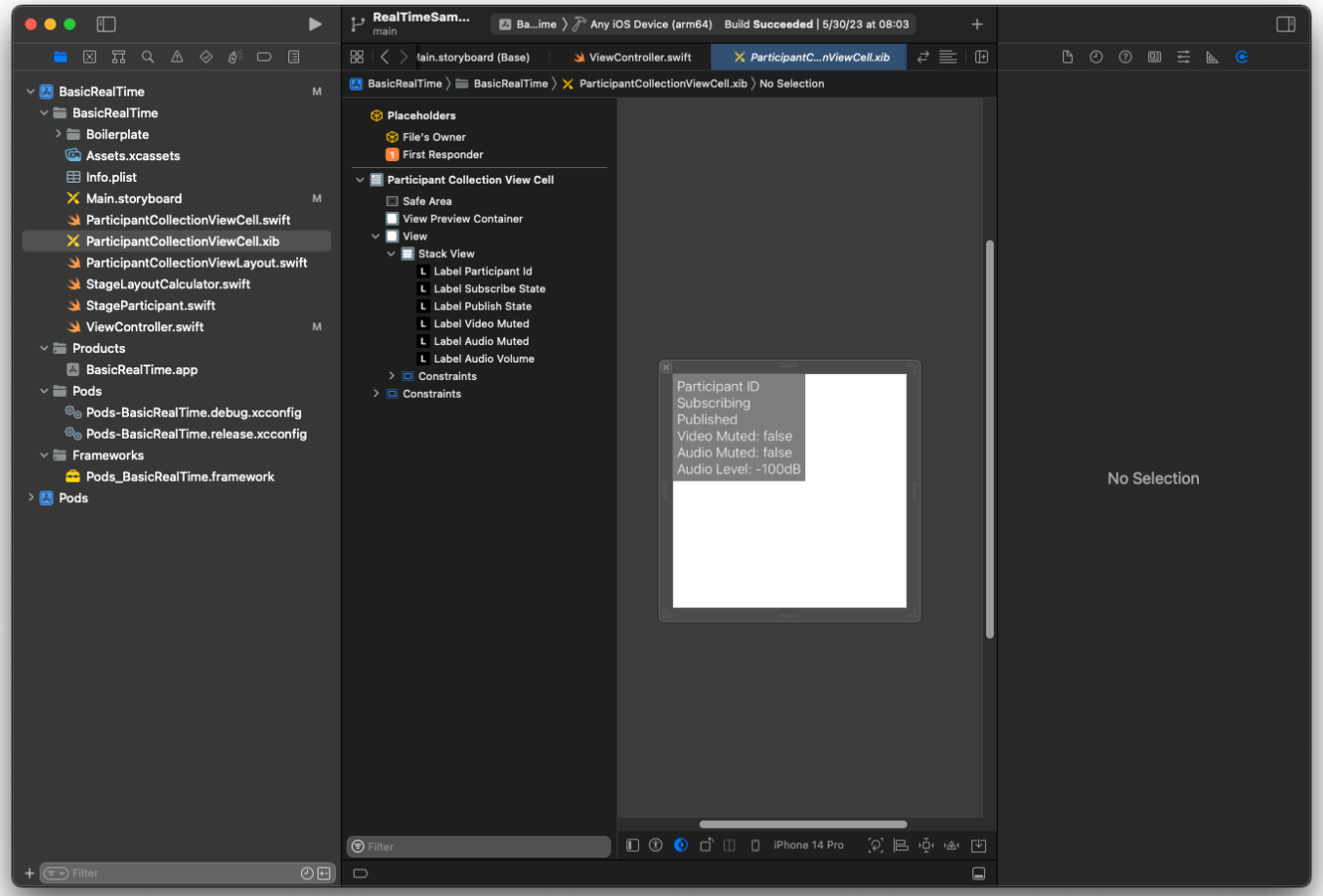
現在，建立要在其中轉譯參與者的 `UICollectionViewCell` 子類別。首先，建立一個新的 Cocoa Touch 類別檔案：



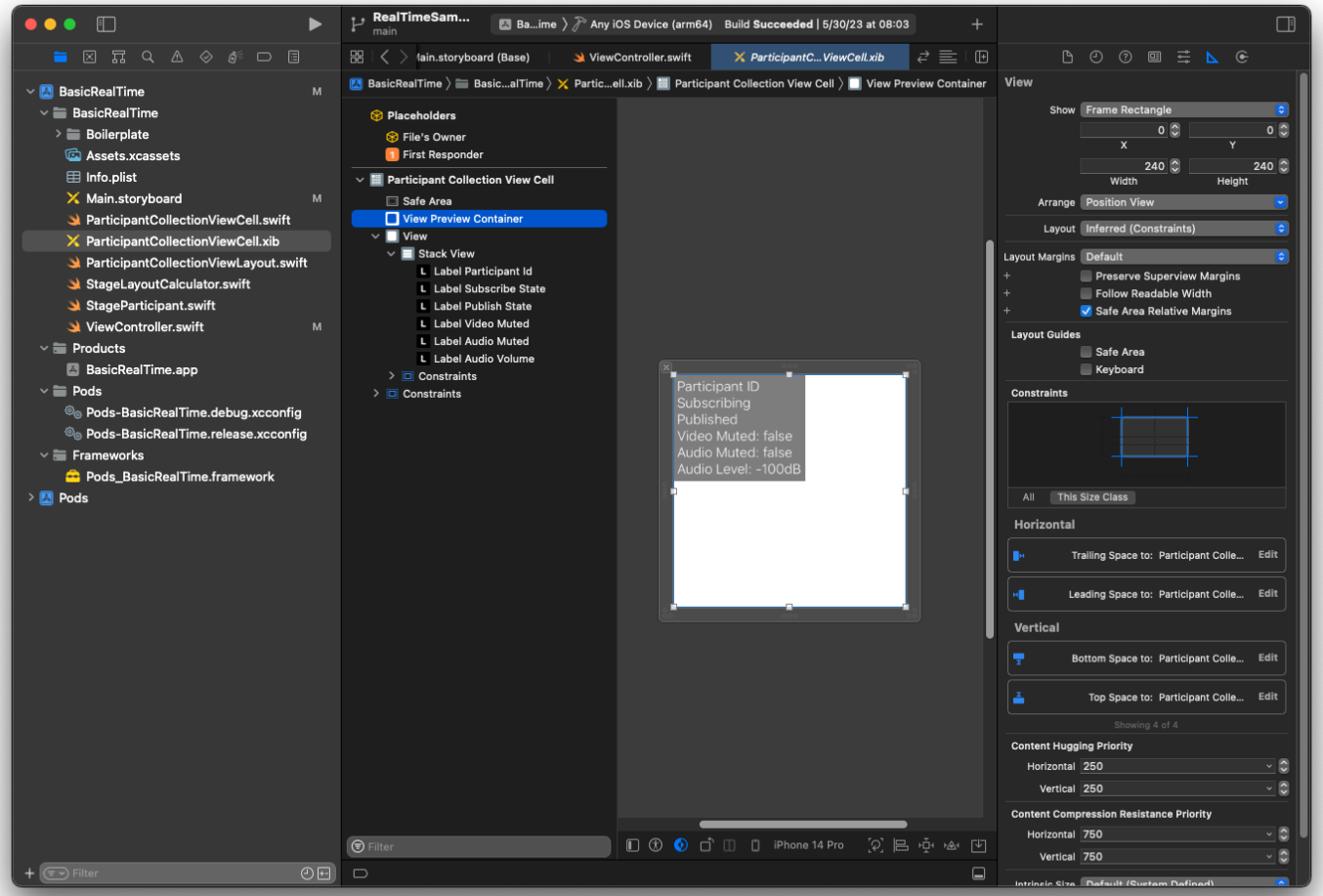
將其命名為 `ParticipantUICollectionViewCell`，然後使其成為 Swift 中 `UICollectionViewCell` 的子類別。再次從 Swift 檔案開始，建立要連結的 `@IBOutlets`：

```
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!
```

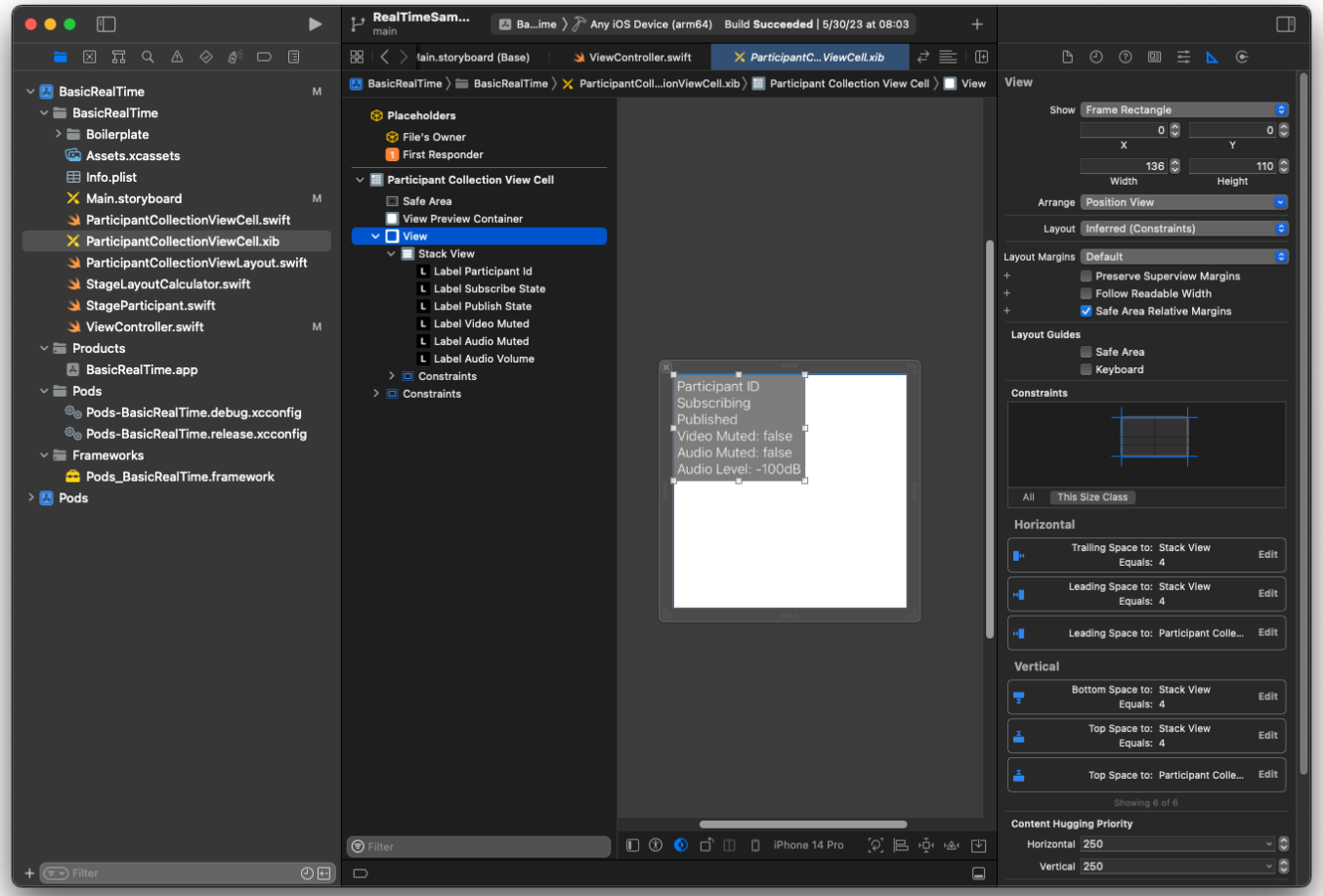
在關聯的 XIB 檔案中，建立此檢視階層：



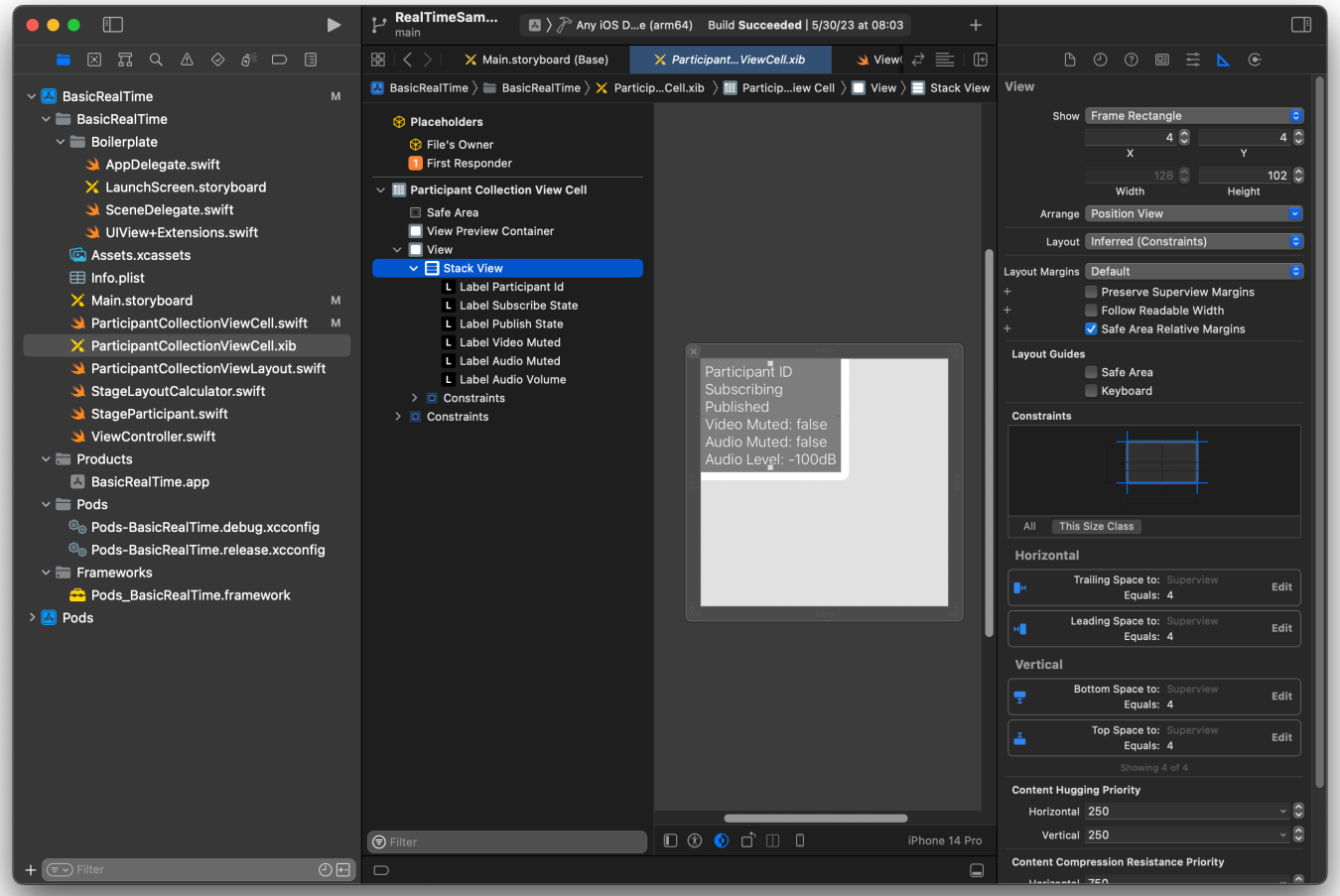
對於 AutoLayout，我們將再次修改三個視圖。第一個檢視是檢視預覽容器。將尾隨、前導、頂部和底部設定為參與者集合檢視儲存格。



第二個檢視是檢視。將前導和頂部設定為參與者集合檢視儲存格，並將該值變更為 4。



第三個檢視是堆疊檢視。將尾隨、前導、頂部和底部設定為父檢視，並將該值變更為 4。



許可和閒置計時器

回到 `ViewController`，將停用系統閒置計時器，以免在使用應用程式時，裝置進入睡眠狀態：

```
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
}
```

接著，請求系統的攝影機和麥克風許可：

```
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)
    }
}
```

應用程式狀態

需要設定 collectionViewParticipants 以及之前建立的配置檔案：

```
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")
}
```

為了代表每個參與者，我們建立了一個稱為 StageParticipant 的簡單結構。這可包含在 ViewController.swift 檔案中，或建立新的檔案。

```
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
    }
}
```

為了追蹤這些參與者，我們將其陣列做為私有屬性保留在 ViewController 中:

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

此屬性將用於支援之前從分鏡腳本連結的 UICollectionViewDataSource：

```
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'")
        }
    }

}
```

若要在加入階段之前查看您自己的預覽，我們會立即建立本機參與者：

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

這會導致在應用程式執行後，代表本機參與者立即轉譯參與者儲存格。

使用者希望能夠在加入階段之前看到自己，因此，接下來我們實作 `setupLocalUser()` 方法，可從稍早的許可處理程式碼中呼叫。將攝影機和麥克風參考做為 `IVSLocalStageStream` 物件來存放。

```
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)
}
```

在這裡透過 SDK 找到了裝置的攝影機和麥克風，並將其存放在本機 streams 物件中，然後將第一個參與者 (我們之前建立的本機參與者) 的 streams 陣列指派到 streams。最後，呼叫 index 為 0，以及 changeType 為 updated 的 participantsChanged。該函數是透過良好的動畫更新 UICollectionView 的輔助程式函數。這是呈現的外觀：

```
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)])
    }
}
```

別擔心 cell.set；稍後會處理，這就是我們將根據參與者轉譯儲存格內容的地方。

ChangeType 是一個簡單的列舉：

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

最後，想要追蹤階段是否已建立連線。使用一個簡單的 bool 來追蹤，在其自行更新時，會自動更新 UI。

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

```
    }
}
```

實作階段 SDK

以下是三個以即時功能為基礎的核心概念：階段、策略和轉譯器。設計目標是盡可能減少打造工作產品所需的用戶端邏輯數量。

智能車 StageStrategy

`IVSStageStrategy` 實作很簡單：

```
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
    }
}
```

總而言之，只有在發布切換按鈕處於「開啟」位置時才會發布，如果要發布，我們會發布之前收集的串流。最後，針對此範例，始終會訂閱其他參與者，同時接收其音訊和影片。

智能車 StageRenderer

`IVSStageRenderer` 實作也非常簡單，但要考慮包含相當多程式碼的函數數量。當 SDK 通知我們有關參與者的變更時，此轉譯器中的一般方法是更新我們的 `participants` 陣列。在某些情況下，對於本機參與者的處理方式有所不同，因為我們已決定自行管理他們，以便在他們加入之前查看其攝影機預覽。

```
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)
    }
}
```

此程式碼使用延伸，將連線狀態轉換為易讀的文字：

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

```
}
```

實作自訂使用者介面 CollectionViewLayout

配置不同數量的參與者可能很複雜。您希望他們佔用整個父檢視的框架，但不想單獨處理每個參與者配置。為了簡化這一點，我們將逐步實作 UICollectionViewLayout。

建立另一個新檔案 ParticipantCollectionViewLayout.swift，這應當會延伸 UICollectionViewLayout。此類別會使用另一個稱為 StageLayoutCalculator 類別，我們很快會介紹。該類別會接收針對每個參與者計算的影格值，然後產生必要的 UICollectionViewLayoutAttributes 物件。

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

更重要的是 StageLayoutCalculator.swift 類別。設計此類別的目的是，根據以流程為基礎的資料列/資料欄配置中的參與者人數，來計算每個參與者的影格。每個資料列的高度與其他資料列相同，但每個資料列的寬度可能有所差異。請參閱 layouts 變數上方的程式碼註解，描述了如何自訂此行為。

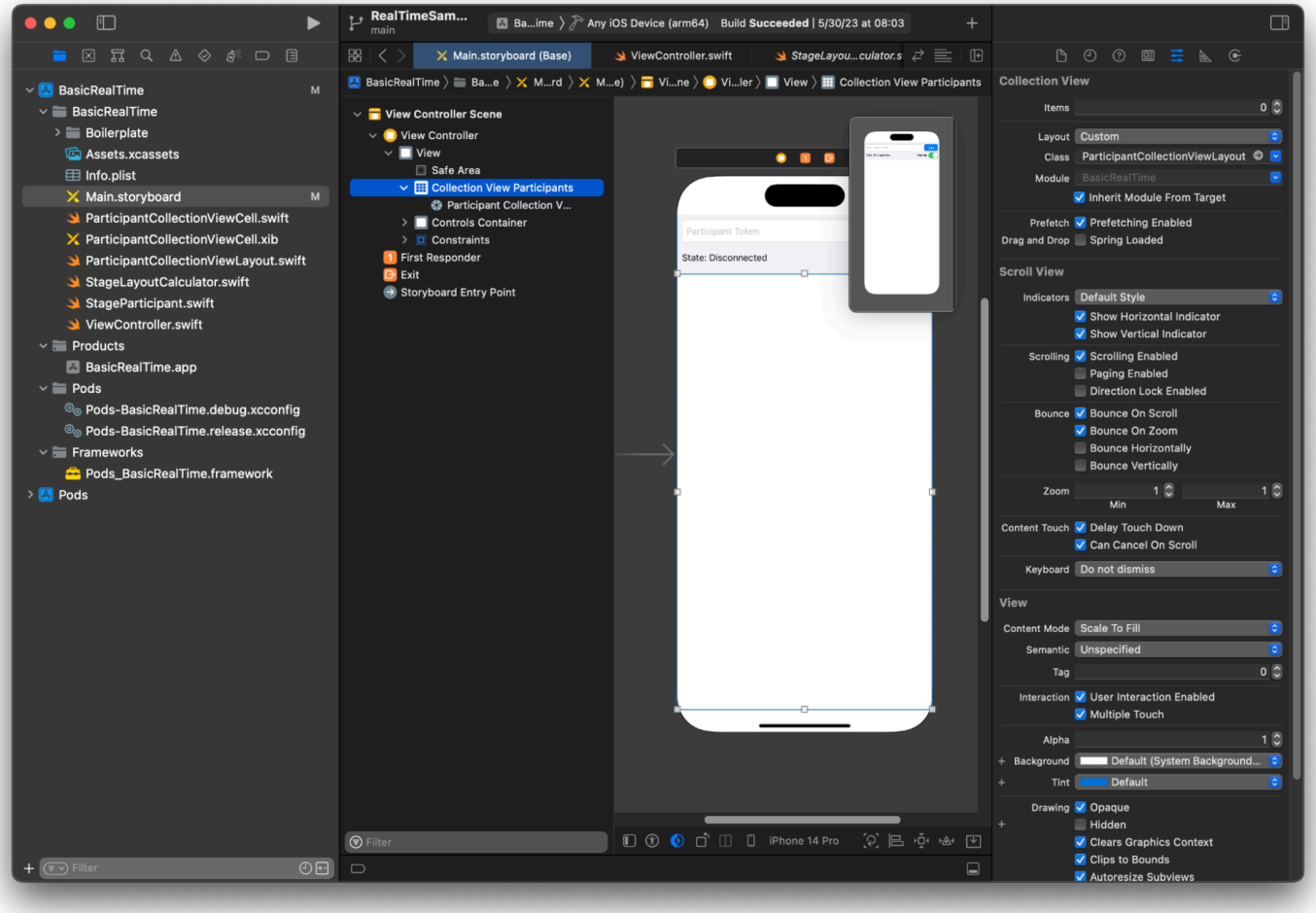
```
import Foundation
import UIKit

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
    }
```

```
}
```

回到 Main.storyboard，請務必將 UICollectionView 的配置類別設定為我們剛才建立的類別：



掛接 UI 動作

我們即將結束，只需建立幾個 IBActions。

首先，將處理加入按鈕。其反應因 connectingOrConnected 的值而異。一旦建立連線後，便會離開階段。如果中斷連線，則會從權杖 UITextField 讀取文字，並建立具有該文字的新的 IVSStage。然後，新增 ViewController 做為 strategy、errorDelegate 和用於 IVSStage 的轉譯器，最後，我們以非同步方式加入階段。

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

需要掛接的另一個 UI 動作是發布切換按鈕：

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

轉譯參與者

最後，我們需要將從 SDK 接收的資料，轉譯到之前建立的參與者儲存格上。我們已經完成了 UICollectionView 邏輯，因此只需實作 ParticipantCollectionViewCell.swift 中的 set API。

從新增 empty 函數開始，然後逐步實作：

```
func set(participant: StageParticipant) {

}
```

首先，處理簡易狀態、參與者 ID、發布狀態和訂閱狀態。對於這些，只需直接更新 UILabels：

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

發布和訂閱列舉的文字屬性源自本機延伸：

```
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()
        }
    }
}
```

接著，更新音訊和影片靜音狀態。為取得靜音狀態，需要從 `streams` 陣列中找到 `IVSImageDevice` 和 `IVSAudioDevice`。為優化效能，會記住上次連接的裝置。

```
// This belongs outside `set(participant:)`
private var registeredStreams: Set<IVSStageStream> = []
private var imageDevice: IVSImageDevice? {
    return registeredStreams.lazy.compactMap { $0.device as? IVSImageDevice }.first
}
private var audioDevice: IVSAudioDevice? {
    return registeredStreams.lazy.compactMap { $0.device as? IVSAudioDevice }.first
}

// This belongs inside `set(participant:)`
```

```
let existingAudioStream = registeredStreams.first { $0.device is IVSAudioDevice }
let existingImageStream = registeredStreams.first { $0.device is IVSImageDevice }

registeredStreams = Set(participant.streams)

let newAudioStream = participant.streams.first { $0.device is IVSAudioDevice }
let newImageStream = participant.streams.first { $0.device is IVSImageDevice }

// `isMuted != false` covers the stream not existing, as well as being muted.
labelVideoMuted.text = "Video Muted: \(newImageStream?.isMuted != false)"
labelAudioMuted.text = "Audio Muted: \(newAudioStream?.isMuted != false)"
```

最後，要轉譯 imageDevice 的預覽，並顯示 audioDevice 的音訊統計資料:

```
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"
}
```

我們需要建立的最後一個函數是 updatePreview()，這會將參與者的預覽新增至檢視：

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

上面使用了 UIView 上的輔助程式函數，使內嵌子檢視變得更容易：

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

監控 Amazon IVS 即時串流

什麼是階段工作階段？

階段工作階段會在第一個參與者加入階段時開始，並在最後一位參與者停止發布至階段的幾分鐘後結束。階段工作階段透過將事件和參與者分離為短期工作階段，來協助對長期階段進行偵錯。

檢視階段工作階段和參與者

主控台說明

1. 開啟 Amazon IVS 主控台。

   (您也可以透過 AWS 管理主控台來存取 Amazon IVS 主控台。)
2. 在導覽窗格中，選擇階段。(如果導航窗格已折疊，請先選擇漢堡圖示將其展開。)
3. 選擇要前往其詳細資訊頁面的階段。
4. 向下捲動頁面，直到看到階段工作階段部分，然後選取階段工作階段以檢視其詳細資訊頁面。
5. 若要檢視工作階段中的參與者，請向下捲動直到您看到參與者區段，然後選取參與者以檢視其詳細資訊頁面，包括 Amazon CloudWatch 指標的圖表。

檢視參與者的事件

當階段中參與者的狀態發生變更 (例如加入階段，或嘗試發布至階段時發生錯誤) 時，則會傳送事件。並非所有錯誤都會導致事件；例如，用戶端網路錯誤和權杖簽章錯誤不會做為事件傳送。若要在用戶端應用程式中處理這些錯誤，請使用 IVS 廣播 SDK。

主控台說明

1. 依照上述指示導覽至參與者詳細資訊頁面。
2. 向下捲動，直到看到事件部分。這會顯示參與者事件的排序清單。如需有關為參與者發出的事件詳細資訊，請參閱搭配使用 Amazon EventBridge 與 Amazon IVS。

CLI 說明

使用 AWS CLI 存取階段工作階段事件是進階選項，需要您先在機器下載並設定 CLI。如需詳細資訊，請參閱 AWS Command Line Interface 使用者指南。

1. 列出階段工作階段以查找階段工作階段：

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

2. 列出階段工作階段的參與者以查找參與者：

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

3. 列出階段工作階段和參與者的事件：

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

以下是回應 `list-participant-events` 呼叫的範例：

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

存取 CloudWatch 指標

若要使用 CloudWatch，需要以下 IVS 廣播 SDK 版本：Web 1.5.0 或以上、Android 1.12.0 或以上，或 iOS 1.12.0 或以上。

CloudWatch 主控台說明

1. 前往 https://console.aws.amazon.com/cloudwatch/ 開啟 CloudWatch 主控台。
2. 在側邊導覽中，展開 Metrics (指標) 下拉式選單，然後選取 All metrics (所有指標)。
3. 在 Browse (瀏覽)索引標籤上，使用左側無標籤的下拉式清單，選取建立頻道的「主要」區域。有關區域的更多資訊，請參閱全球解決方案、區域控制。如需支援的區域清單，請參閱 AWS 一般參考中的 Amazon IVS 頁面。
4. 在 Browse (瀏覽) 索引標籤底部，選取 IVSRealTime 命名空間。
5. 執行以下任意一項：
   a. 在搜尋列中，輸入您的資源 ID (ARN 的一部分，`arn:::ivs:stage/<resource id>`)。

      然後選取 IVSRealTime > 階段指標。

   b. 如果 IVSRealTime 在 AWS 命名空間下顯示為可選取的服務，請選取它。如果您使用 Amazon IVS 即時串流功能 並且它正在傳送指標到 Amazon CloudWatch，則將列出它。(如果未列出 IVSRealTime，則您沒有任何 Amazon IVS 指標。)

      然後根據需要選擇維度分組；可用的維度會列在下方的 CloudWatch 指標中。
6. 選擇要新增到圖表的指標。可用的指標列在 CloudWatch 的指標。

您也可以選取 View in CloudWatch (在 CloudWatch 中檢視) 按鈕，從串流工作階段的詳細資訊頁面存取串流工作階段的 CloudWatch 圖表。

CLI 說明

您也可以使用 AWS CLI 存取指標。這需要在您的機器上先下載並設定 CLI。如需詳細資訊，請參閱 AWS 命令列界面使用者指南。

然後，使用 AWS CLI 存取 Amazon IVS 即時串流功能 指標：

- 在命令提示中，執行：

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

如需詳細資訊，請參閱 Amazon CloudWatch 使用者指南中的使用 Amazon CloudWatch 指標。

CloudWatch 指標：IVS 即時串流

Amazon IVS 在 AWS/IVSRealTime 命名空間中提供以下指標。

若要使用 CloudWatch 指標，必須使用網路廣播 SDK 1.5.2 或更新版本。

此維度可以具有以下有效值：

- `Stage` 維度是資源 ID (ARN 的一部分,`arn:::stage/<resource id>`)。
- `Participant` 維度是 `participantID`。
- `SimulcastLayer` 為 "hi"、"mid"、"low" 或 "no-rid" (`MediaType` 為「影片」) 或「已停用」(`MediaType` 為「音訊」)。此值也可以為空白。
- `MediaType` 維度為「視訊」或「音訊」(字串)。

指標	維度	描述
`DownloadPacketLoss`	`Stage`	每個範例均代表指定訂閱用戶從 IVS 伺服器下載時遺失的封包百分比。 單位：百分比 有效統計數字：平均值、最大值、最小值 – 設定間隔內擷取封包遺失的平均數、最大數或最小數 (分別)
`DownloadPacketLoss`	`Stage,Participant`	依參與者篩選 `DownloadPacketLoss` ，針對也是發布者的訂閱用戶。範例代表訂閱用戶從 IVS 伺服器下載時遺失的封包百分比。只有當參與者也是發布者時，才會發出範例。 單位：百分比

指標	維度	描述
		有效統計數字：平均值、最大值、最小值 – 設定間隔內捨棄影格速率的平均數、最大數或最小數 (分別)
DroppedFrames	Stage	每個範例都代表指定訂閱用戶捨棄的影格百分比。 單位：百分比 有效統計數字：平均值、最大值、最小值 – 設定間隔內捨棄影格速率的平均數、最大數或最小數 (分別)
DroppedFrames	Stage,Participant	依參與者篩選 DroppedFrames ，針對也是發布者的訂閱用戶。範例代表訂閱參與者與階段中的所有發布者之間捨棄的影格百分比。只有當參與者也是發布者時，才會發出範例。 單位：百分比 有效統計數字：平均值、最大值、最小值 – 設定間隔內捨棄影格速率的平均數、最大數或最小數 (分別)
PublishBitrate	Stage	發出的範例代表指定發行者傳送視訊和音訊資料的總速率 (跨所有同步廣播層加總)。 單位：位元/秒 有效統計數字：平均值、最大值、最小值 – 設定間隔內位元速率的平均數、最大數或最小數 (分別)
PublishBitrate	Stage, Participant, SimulcastLayer, MediaType	依參與者、同步廣播層和媒體類型篩選 PublishBitrate 。同步廣播層 ID 是由廣播 SDK 設定。停用同步廣播時，此層 ID 將設定為「停用」。媒體類型為視訊或音訊。 單位：位元/秒 有效統計數字：平均值、最大值、最小值 – 設定間隔內位元速率的平均數、最大數或最小數 (分別)

指標	維度	描述
Publishers	Stage	發布至階段的參與者人數。 單位：計數 有效統計：平均值、最大值、最小值
PublishResolution	Stage, Participant, SimulcastLayer, MediaType	跨越框架寬度或高度較小的像素數。例如，對於尺寸為 1920x1080 的橫向框架，PublishResolution 為 1080。對於尺寸為 720x1280 的縱向框架，PublishResolution 為 720。 單位：計數 有效統計：平均值、最大值、最小值
SubscribeBitrate	Stage	發出的範例代表指定訂閱用戶接收視訊和音訊資料的總速率。 單位：位元/秒 有效統計數字：平均值、最大值、最小值 – 設定間隔內位元速率的平均數、最大數或最小數 (分別)
SubscribeBitrate	Stage,Participant, MediaType	依參與者篩選 SubscribeBitrate ，針對也是發布者的訂閱用戶。範例代表指定訂閱用戶接收指定 MediaType 的位元速率。只有在訂閱參與者發布時才會發出範例。 單位：位元/秒 有效統計數字：平均值、最大值、最小值 – 設定間隔內位元速率的平均數、最大數或最小數 (分別)
Subscribers	Stage	已訂閱階段的參加者人數。請注意，主動發布及訂閱的參與者同時被視為發布者和訂閱用戶。 單位：計數 有效統計：平均值、最大值、最小值

IVS 廣播 SDK (即時串流)

Amazon Interactive Video Services (IVS) 即時串流廣播 SDK 適用於使用 Amazon IVS 建置應用程式的開發人員。此 SDK 的設計目的是利用 Amazon IVS 架構，並持續使用 Amazon IVS 的改善之處和新功能。作為原生廣播 SDK，其設計目的是將對您的應用程式和使用者存取應用程式的裝置的效能影響降至最低。

請注意，廣播 SDK 用於傳送和接收影片；也就是說，您對主持人和觀眾使用相同的 SDK。無需單獨的播放器 SDK。

您的應用程式可以利用 Amazon IVS 廣播 SDK 的主要功能：

- 高品質串流 — 廣播 SDK 支援高品質串流。從攝影機擷取影片並以最高 720p 的速度對其進行編碼。
- 自動調整位元速率 — 智慧型手機使用者處於移動狀態，網路狀況可能在整個廣播過程中變更。Amazon IVS 廣播 SDK 會自動調整影片位元速率，以適應不斷變化的網路狀況。
- 縱向和橫向支援 — 無論使用者如何手持裝置，影像都會在右側向上顯示並正確縮放。廣播 SDK 支援縱向和橫向畫布大小。當使用者的裝置旋轉方向與影片設定的方向不同時，它會自動管理長寬比。
- 安全串流 — 使用 TLS 加密使用者的廣播，保護串流的安全。
- 外部音訊裝置 — Amazon IVS 廣播 SDK 支援音訊插孔，USB 和藍牙 SCO 外接麥克風。

平台需求：

原生平台

平台	支援的版本
Android	9.0 及更高版本 – 請注意，客戶可以使用 5.0 版進行建置，但無法使用即時串流功能。
iOS	14 版及更高版本

IVS 至少支援 4 個主要的 iOS 版本和 6 個主要的 Android 版本。我們目前的版本支援可能會超過這些最低限度。客戶至少會提前 3 個月透過 SDK 版本備註收到通知，知悉某個主要版本不再受支援。

桌面瀏覽器

瀏覽器	支援的平台	支援的版本
Chrome	Windows、macOS	兩個主要版本 (目前版本和最新的先前版本)
Firefox	Windows、macOS	兩個主要版本 (目前版本和最新的先前版本)
Edge	Windows 8.1 及更新版本	兩個主要版本 (目前版本和最新的先前版本) 排除邊緣舊版
Safari	macOS	兩個主要版本 (目前版本和最新的先前版本)

移動瀏覽器 (iOS 和 Android)

瀏覽器	支援的平台	支援的版本
Chrome	iOS、Android	兩個主要版本 (目前版本和最新的先前版本)
Firefox	Android	兩個主要版本 (目前版本和最新的先前版本)
Safari	iOS	兩個主要版本 (目前版本和最新的先前版本)

已知限制

- 在所有行動裝置上，我們都不建議同時發布/訂閱四位或更多參與者，因為存在影片成品和黑畫面的問題。如果您需要更多參與者，請設定純音訊發布和訂閱。
- 考慮到效能和潛在的當機問題，我們不建議您合成舞台並將其廣播到 Android 行動 Web 上的頻道。如果需要廣播功能，請整合 IVS 即時串流 Android 廣播 SDK。

Webview

Web 廣播 SDK 不提供對 Webview 或類似 Web 之環境 (電視、主控台等) 的支援。如需行動裝置實作，請參閱 Android 版和 iOS 版即時串流廣播 SDK 指南。

必要的裝置存取權

廣播 SDK 需要存取裝置的攝影機和麥克風，包括裝置內建的攝影機和麥克風，以及透過藍牙、USB 或音訊插孔連接的攝影機和麥克風。

支援

廣播 SDK 會持續改善。請參閱可用版本的 Amazon IVS 版本備註以及已修正的問題。如果適當，請在聯絡支援部門之前，先更新您的廣播 SDK 版本，並查看是否可以解決您的問題。

版本控制

Amazon IVS 廣播 SDK 使用語意版本控制。

對於此討論，假設：

- 最新版本為 4.1.3 版。
- 先前主要版本的最新版本為 3.2.4 版。
- 版本 1.x 的最新版本為 1.5.6 版。

回溯相容的新功能會新增為最新版本的次要版本。在這種情況下，下一組新功能將被新增為 4.2.0 版。

回溯相容的次要錯誤修正會新增為最新版本的修補程式版本。在這裡，下一組小錯誤修復將被新增為 4.1.4 版。

回溯相容、主要錯誤修正的處理方式不同；它們會新增至多個版本：

- 最新版本的修補程式版本。在這裡，它為 4.1.4 版。
- 先前次要版本的修補程式版本。在這裡，它為 3.2.5 版。
- 最新版 1.x 版本的修補程式版本。在這裡，它為 1.5.7 版。

主要錯誤修正由 Amazon IVS 產品團隊定義。典型範例包括重要的安全更新以及客戶所需的其他精選修正。

備註：在上面的範例中，發布的版本在不跳過任何數字的情況下遞增 (例如，從 4.1.3 到 4.1.4)。實際上，一個或多個修補程式編號可能會保持在內部並且不需要發行，因此發行的版本可能會從 4.1.3 增加到 4.1.6。

IVS 廣播 SDK：Web 指南 (即時串流)

IVS 即時串流 Web 廣播 SDK 為開發人員提供了在 Web 上建立互動式即時體驗的工具。此 SDK 適用於使用 Amazon IVS 建置 Web 應用程式的開發人員。

Web 廣播 SDK 讓參與者能夠傳送和接收影片。SDK 支援下列操作：

- 加入階段
- 將媒體發布給階段中的其他參與者
- 訂閱階段中其他參與者的媒體
- 管理和監控發布到階段的影片和音訊
- 取得每個對等連線的 WebRTC 統計資料
- IVS 低延遲串流 Web 廣播 SDK 的所有操作

最新版本的網路廣播 SDK：1.8.0 (版本說明)

參考文件：如需 Amazon IVS 網路廣播開發套件中最重要方法的相關資訊，請參閱 https://aws.github.io/ amazon-ivs-web-broadcast /文檔/ SDK 參考資料。請確定已選取最新版本的 SDK。

範本程式碼：透過以下範例可快速了解 SDK：

- HTML 和 JavaScript
- 反應

平台需求：支援平台的清單請參閱 Amazon IVS 廣播 SDK。

開始

匯入

多位主持人適用的建構區塊所在的命名空間與根廣播模組不同。

使用指令碼標籤

使用相同指令碼會匯入後，下方範例定義的類別和列舉便可以在全域物件 IVSBroadcastClient 上找到：

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

使用 NPM

類別、列舉和類型也可以從套件模組導入：

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

請求權限

您的應用程式必須請求許可，才能存取使用者的攝影機和麥克風，而您必須使用 HTTPS 為其提供服務。(這不限於 Amazon IVS；任何需要存取攝影機和麥克風的網站都必須如此。)

以下範例函數顯示如何請求和擷取音訊及影片裝置的許可：

```
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.');
   }
```

```
}
```

如需詳細資訊，請參閱權限 API 和 MediaDevices. getUserMedia()。

列出可用裝置

若要查看哪些裝置可以擷取，請查詢瀏覽器的 MediaDevices.enumerateDevices () 方法：

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

從設備 MediaStream 中檢索

取得可用裝置清單後，您就可以從任意數量的裝置中擷取串流。例如，您可以使用 `getUserMedia()` 方法來擷取攝影機的串流。

如果想指定要從哪個裝置擷取串流，您可以在媒體限制條件的 `audio` 或 `video` 區段中明確設定 `deviceId`。或者，您可以省略 `deviceId` 並讓使用者從瀏覽器提示中選取其裝置。

您也可以使用 `width` 和 `height` 限制條件指定理想的攝影機解析度。(在此處閱讀有關這些限制條件的更多資訊。) SDK 會自動套用與您最大廣播解析度相對應的最大寬度和高度限制條件；不過，最好還是自行套用這些限制條件，以確保在將來源新增至 SDK 之後，來源長寬比不會遭到變更。

若要進行即時串流，請確定媒體限制為 720p 解析度。具體而言，寬度`getUserMedia`和高度的和`getDisplayMedia`約束值在相乘時不得超過 921600 (1280*720)。

```
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 },
});
```

發布與訂閱

概念

以下是三個以即時功能為基礎的核心概念：階段、策略 和 事件。設計目標是盡可能減少打造工作產品所需的用戶端邏輯數量。

階段

Stage 類別是主持人應用程式和 SDK 之間的主要交互點。它代表階段本身，用於加入和離開階段。建立和加入階段需有效且未過期的控制平面權杖字串 (表示為 token)。加入和離開階段並不難：

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

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

stage.leave();
```

策略

StageStrategy 介面為主持人應用程式提供了將所需階段狀態傳送至 SDK 的管道。您必須實作以下三項函數：shouldSubscribeToParticipant、shouldPublishParticipant、和 stageStreamsToPublish。以下將討論所有內容。

若要使用已定義的策略，請將其傳送給 Stage 建構函數。以下是使用策略將參與者的網路攝影機發布到階段並訂閱所有參與者的完整應用程式範例。後續幾節會詳細說明各項必要策略函數的用途。

```
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();
```

訂閱參與者

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

遠端參與者加入階段時，SDK 會向主持人應用程式查詢該參與者所需的訂閱狀態。選項包括 NONE、AUDIO_ONLY 和 AUDIO_VIDEO。傳回此函數的值時，主持人應用程式不需要擔心發布狀態、目前的訂閱狀態或階段連線狀態。若傳回 AUDIO_VIDEO，SDK 會等到遠端參與者發布時才會訂閱，然後在整個程序中透過發出事件來更新主持人應用程式。

以下是實作範例：

```
const strategy = {

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

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

對於一律希望所有參與者互相看到彼此的主持人應用程式 (例如影片聊天應用程式)，這是此函數的完整實作程序。

您也可以採用更進階的實作方式。在 ParticipantInfo 上使用 userInfo 屬性，以根據伺服器提供的屬性選擇性訂閱參與者：

```
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
}
```

這可以用來建立一個階段，版主可以在不會被看到或聽到自己聲音的情況下監控所有訪客。主持人應用程式可以使用其他商業邏輯，讓版主看到彼此，但仍維持訪客看不到他們的狀態。

發布

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

連線到階段後，SDK 會查詢主持人應用程式，看看特定參與者是否應發布。系統僅會根據提供的字符為具有發布許可的本機參與者調用此函數。

以下是實作範例：

```
const strategy = {

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

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

這是針對一個使用者總是想發布內容的標準影片聊天應用程式。他們可以靜音和取消靜音其音訊和影片內容，以立即隱藏起來，或看到/聽見內容。(他們也可以使用發布/取消發布，但這種方式速度較慢。建議在需經常變更可見性的使用案例中使用靜音/取消靜音。)

選擇要發布的串流

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

發布時，這會用來決定應發布哪些音訊和影片串流。稍後會在發布媒體串流中進行詳細說明。

更新策略

策略應處於動態狀態：從上述任何函數返回的值可以隨時進行修改。例如，若主持人應用程式在終端使用者按下按鈕前都不想發布，您可以從 `shouldPublishParticipant` 傳回一個變數 (例如 `hasUserTappedPublishButton`)。當該變數根據終端使用者的互動而變更時，請呼叫 `stage.refreshStrategy()` 向 SDK 傳送信號，表示它應查詢策略中的最新值，並僅套用已變更的項目。若 SDK 發現 `shouldPublishParticipant` 值已變更，它便會開始進行發布。若 SDK 查詢後所有函數傳回與之前相同的值，則 `refreshStrategy` 呼叫將不會對階段進行修改。

若 shouldSubscribeToParticipant 傳回的值從 AUDIO_VIDEO 變更為 AUDIO_ONLY，則系統會針對傳回值已變更的所有參與者移除影片串流 (若之前存有影片串流)。

一般而言，階段會採用策略，以最有效率的方式套用先前與目前策略之間的差異，主持人應用程式不必擔心正確進行管理所需的所有狀態。因此，請將呼叫 stage.refreshStrategy() 視為低成本的操作，因為除非策略發生變化，否則它什麼都不會執行。

事件

Stage 執行個體是一個事件觸發器。使用 stage.on() 時，系統會將階段的狀態傳送給主持人應用程式。主持人應用程式的 UI 更新通常可以完全由事件提供支援。事件如下所示：

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

針對大多數這些事件提供了相應的 ParticipantInfo。

事件提供的資訊應該不會對策略的傳回值造成影響。例如，呼叫 STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED 時，shouldSubscribeToParticipant 的傳回值應該不會變更。若主持人應用程式想要訂閱特定參與者，則無論該參與者的發布狀態為何，它都應傳回所需的訂閱類型。SDK 負責確保根據階段狀態，在正確的時間點執行策略的所需狀態。

發布媒體串流

本機裝置 (例如麥克風和攝影機) 的擷取步驟與上述 MediaStream 從裝置擷取相同的步驟。在範例中，我們會使用 MediaStream 來建立 SDK 用來發布的 LocalStageStream 物件清單：

```
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
    }
}
```

發布螢幕共用

除了使用者的網路攝影機外，應用程式通常需要發布螢幕共用。發布螢幕共用需額外建立一個具有不重複字符的 Stage。

```
// 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();
```

顯示和移除參與者

訂閱完成後，您會透過 STAGE_PARTICIPANT_STREAMS_ADDED 事件收到 StageStream 物件陣列。該事件還會為您提供參與者的資訊，在您顯示媒體串流時提供協助：

```
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));
})
```

當參與者停止發布或取消訂閱串流時，系統會呼叫 STAGE_PARTICIPANT_STREAMS_REMOVED 函數，並傳回遭移除的串流。主持人應用程式應將此視為從 DOM 中移除參與者影片串流的信號。

STAGE_PARTICIPANT_STREAMS_REMOVED 會在串流可能遭移除的所有情況下調用，其中包括：

- 遠端參與者停止發布。
- 本機裝置取消訂閱，或將訂閱從 AUDIO_VIDEO 變更為 AUDIO_ONLY。
- 遠端參與者離開階段。
- 本機參與者離開階段。

由於 STAGE_PARTICIPANT_STREAMS_REMOVED 會在所有情況下調用，因此在遠端或本機離開操作期間，不需要使用自訂商業邏輯從 UI 移除參與者。

靜音和取消靜音媒體串流

LocalStageStream 物件具備控制是否將串流靜音的 setMuted 函數。此函數可以在從 stageStreamsToPublish 策略函數傳回之前或之後在串流上呼叫。

重要：如果呼叫 refreshStrategy 後由 stageStreamsToPublish 傳回新的 LocalStageStream 物件執行個體，則新串流物件的靜音狀態會套用至階段。建立新 LocalStageStream 執行個體時請務必小心，以確保維持預期的靜音狀態。

監控遠端參與者媒體靜音狀態

當參加者變更其影片或音訊的靜音狀態時，會以已變更的串流清單觸發 STAGE_STREAM_MUTE_CHANGED 事件。使用 StageStream 上的 isMuted 屬性來據此更新您的 UI：

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

此外，您可以查看StageParticipantInfo有關音頻或視頻是否靜音的狀態信息：

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

取得 WebRTC 統計資料

若要取得發布串流或訂閱串流的最新 WebRTC 統計資料，請在 StageStream 上使用 getStats。這是一種非同步方法，讓您可以透過等待或鏈結承諾來擷取統計資料。結果為含有所有標準統計資料的字典 RTCStatsReport。

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

最佳化媒體

建議您為 getUserMedia 和 getDisplayMedia 呼叫設下以下限制，以獲得最佳效能：

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

```
        height: { ideal: 720 },
        framerate: { ideal: 30 },
    },
};
```

您可以透過傳遞至 LocalStageStream 建構函數的其他選項進一步限制媒體：

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

在上述程式碼中：

- minBitrate 設定瀏覽器預期應使用的最小位元速率。然而，低複雜度影片串流可能會使編碼器低於此位元速率。
- maxBitrate 設定瀏覽器預期應不超過此串流的最大位元速率。
- maxFramerate 設定瀏覽器預期應不超過此串流的最大影格率。
- simulcast 選項僅適用於 Chromium 瀏覽器。它可傳送串流的三個轉譯層。
  - 這讓伺服器能夠根據其網路限制，選擇要傳送給其他參與者的轉譯。
  - 連同 maxBitrate 及/或 maxFramerate 值一起指定 simulcast 時，預期最高轉譯層會考慮設定這些值，前提條件是 maxBitrate 不低於內部 SDK 第二高層 900 kbps 的預設 maxBitrate 值。
  - 如果相較於第二高層的預設值，指定的 maxBitrate 太低，則會停用 simulcast。
  - 若未透過將 shouldPublishParticipant 傳回 false、呼叫 refreshStrategy、將 shouldPublishParticipant 傳回 true，以及再次呼叫 refreshStrategy 的組合動作來重新發布媒體，則無法開啟和關閉 simulcast。

取得參與者屬性

如果您在 CreateParticipantToken 端點要求中指定屬性，您可以在 StageParticipantInfo 屬性中看到屬性：

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

處理網路問題

當本機裝置的網路連線中斷時，SDK 會在內部嘗試重新連線，無需使用者採取任何動作。SDK 在部分情況下會執行失敗，這時就需要使用者採取動作。

階段的狀態大致上可以透過 STAGE_CONNECTION_STATE_CHANGED 事件來進行處理：

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

一般而言，若成功加入階段後遇到錯誤，則表示 SDK 連線中斷，且沒有成功重新建立連線。建立新的 Stage 物件，並在網路情況改善時嘗試加入。

將階段廣播到 IVS 頻道

若要廣播階段，請建立一個獨立 IVSBroadcastClient 工作階段，然後按照使用 SDK 進行廣播的一般指示操作 (如上所述)。透過 STAGE_PARTICIPANT_STREAMS_ADDED 公開的 StageStream 清單可用於擷取可應用於廣播串流組合的參與者媒體串流，如下所示：

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

或者，您可以複合階段並將其廣播到 IVS 低延遲通道，以吸引更多受眾。請參閱《IVS 低延遲串流使用者指南》中的在 Amazon IVS 串流上啟用多位主持人。

已知問題和解決方法

- 在不呼叫 `stage.leave()` 的情況下關閉瀏覽器分頁或退出瀏覽器時，使用者仍可在工作階段中以凍結畫面或黑色畫面顯示長達 10 秒的時間。

  解決方法：無。

- Safari 工作階段開始後，工作階段會斷斷續續地向加入的使用者顯示黑色畫面。

  解決方法：重新整理瀏覽器並重新連線工作階段。

- Safari 無法在切換網路後正常復原。

  解決方法：重新整理瀏覽器並重新連線工作階段。

- 開發人員主控台重複出現 `Error: UnintentionalError at StageSocket.onClose` 錯誤。

  解決方法：每個參與者權杖只能建立一個階段。使用相同參與者權杖建立多個 Stage 執行個體時，無論執行個體位於一或多部裝置，都會發生此錯誤。

- 您可能無法維持`StageParticipantPublishState.PUBLISHED`狀態，並且在聆聽`StageEvents.STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED`事件時可能會收到重複的`StageParticipantPublishState.ATTEMPTING_PUBLISH`狀態。

因應措施：呼叫`getUserMedia`或`getDisplayMedia`時，將視訊解析度限制為 720p。具體而言，寬度`getUserMedia`和高度的和`getDisplayMedia`約束值在相乘時不得超過 921600 (1280*720)。

Safari 限制

- 拒絕許可提示需要在作業系統層級的 Safari 網站設定中重設許可。
- Safari 不像 Firefox 或 Chrome，其原本並能有效地偵測所有裝置。例如，其無法偵測到 OBS 虛擬攝影機。

Firefox 限制

- 必須啟用 Firefox 的系統許可，才能進行螢幕共用。啟用後，使用者必須重新啟動 Firefox 才能正常運作；否則，如果權限被視為封鎖，瀏覽器就會擲回NotFoundError例外狀況。
- 缺少 `getCapabilities` 方法。這意味著使用者無法取得媒體軌道的解析度或長寬比。請參閱這個 bugzilla 討論串。
- 缺少數個 `AudioContext` 屬性；例如，延遲和頻道計數。對於想要操作音軌的進階使用者來說，這可能會造成問題。

- 在 MacOS 上，來自 `getUserMedia` 的攝影機供稿的長寬比限制為 4:3。請參閱 bugzilla 討論串 1 和 bugzilla 討論串 2。

- 不支援使用 `getDisplayMedia` 進行音訊擷取。請參閱這個 bugzilla 討論串。
- 螢幕擷取畫面中的影格速率不理想 (大約 15fps？)。請參閱這個 bugzilla 討論串。

行動 Web 限制

- getDisplayMedia在移動設備上不支持屏幕共享。

  解決方法：無。
- 參與者在不呼叫 `leave()` 就關閉瀏覽器時，需要 15-30 秒才能離開。

  解決方法：新增 UI 鼓勵使用者正確中斷連線。
- 背景應用程式會導致發布影片停止。

解決方法：在發布者暫停時顯示 UI 靜態圖像。

- 在 Android 裝置上取消攝影機靜音後，影片影格率會下降約 5 秒鐘。

  解決方法：無。

- 在 iOS 16.0 的旋轉中，影片供稿會延伸。

  解決方法：顯示 UI 概述此已知的作業系統問題。

- 切換音訊輸入裝置時會自動切換音訊輸出裝置。

  解決方法：無。

- 背景化瀏覽器會導致發佈資料流變黑並僅產生音訊。

  解決方法：無。這是出於安全原因。

錯誤處理

本節概述錯誤情況、Web 廣播 SDK 如何向應用程式報告錯誤，以及應用程式在遇到這些錯誤時應執行的動作。錯誤分為四種類別：

```
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
  }
});
```

階段執行個體化錯誤

階段執行個體化不會遠端驗證權杖，但會檢查可在用戶端驗證的基本權杖問題。因此，SDK 會可能擲出錯誤。

格式錯誤的參與者權杖

當階段權杖格式不正確時，就會發生此錯誤。執行個體化階段時，SDK 會擲出錯誤並顯示以下訊息：「剖析階段權杖時發生錯誤」。

動作：建立一個有效權杖，然後重試執行個體化。

加入階段錯誤

這些錯誤可能在最初嘗試加入階段時發生。

階段已刪除

當加入已刪除的階段 (與權杖相關聯) 時，就會發生此錯誤。joinSDK 方法會擲回錯誤訊息：「10 秒 InitialConnectTimedOut 後」。

動作：使用新階段建立一個有效權杖，然後重試加入。

參與者權杖過期

當權杖過期時，就會發生此錯誤。join SDK 方法會擲回錯誤並顯示以下訊息：「權杖過期且失效。」

動作：建立一個新權杖，然後重試加入。

參與者權杖無效或撤銷

當權杖無效或被撤銷/中斷連線時，就會發生此錯誤。joinSDK 方法會擲回錯誤訊息：「10 秒 InitialConnectTimedOut 後」。

動作：建立一個新權杖，然後重試加入。

權杖中斷連線

當階段權杖格式正確但遭 Stages 伺服器拒絕時，就會發生此錯誤。joinSDK 方法會擲回錯誤訊息：「10 秒InitialConnectTimedOut 後」。

動作：建立一個有效權杖，然後重試加入。

初始加入時出現網路錯誤

當 SDK 無法聯絡 Stages 伺服器以建立連線時，就會發生此錯誤。joinSDK 方法會擲回錯誤訊息：「10 秒InitialConnectTimedOut 後」。

動作：等待裝置的連線復原，然後重試加入。

已加入時出現網路錯誤

如果裝置的網路連線中斷，SDK 可能會失去與 Stage 伺服器的連線。您可能會在主控台中看到錯誤，因為 SDK 無法再連線到至後端服務。https://broadcast.stats.live-video.net 的 POST 會失敗。

如果您正在發布和/或訂閱，您會在主控台中看到與嘗試發布/訂閱相關的錯誤。

SDK 會在內部嘗試與指數退避策略重新連線。

動作：等待裝置的連線復原。如要發布或訂閱，請重新整理策略以確保媒體串流重新發布。

發布和訂閱錯誤

發布錯誤：發布狀態

SDK 會在發布失敗時報告 ERRORED。這可能是因為網路狀況或發布者的階段容量達上限所致。

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

動作：重新整理策略以嘗試重新發布媒體串流。

訂閱錯誤

訂閱失敗時 SDK 會報告 ERRORED。這可能是因為網路狀況或訂閱者的階段容量達上限所致。

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

動作:重新整理策略以嘗試新訂閱。

IVS 廣播 SDK：Android 指南 (即時串流)

IVS 即時串流 Android 廣播 SDK 讓參與者能夠在 Android 上傳送和接收影片。

`com.amazonaws.ivs.broadcast` 套件會執行本文件中所述的介面。SDK 支援下列操作：

- 加入階段
- 將媒體發布給階段中的其他參與者
- 訂閱階段中其他參與者的媒體
- 管理和監控發布到階段的影片和音訊
- 取得每個對等連線的 WebRTC 統計資料
- IVS 低延遲串流 Android 廣播 SDK 的所有操作

安卓廣播軟體開發套件的最新版本：1.14.1 (發行說明)

參考文件：有關 Amazon IVS 安卓廣播開發套件中可用的最重要方法的資訊，請參閱參考文件，網址為 https://aws.github.io/amazon-ivs-broadcast-docs/1.14.1/android。

示例代碼：請參閱以下位置的安卓示例存儲庫 GitHub：https://github.com/aws-samples/ amazon-ivs-broadcast-android-示例。

平台需求：Android 9.0 及更高版本。

開始

安裝程式庫

若要將 Amazon IVS Android 廣播程式庫新增到您的 Android 開發環境中，請將該程式庫新增到模組的 `build.gradle` 檔案，如下所示 (適用於 Amazon IVS 廣播 SDK 的最新版本)：

```
repositories {
    mavenCentral()
}

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

將下列許可新增至您的清單檔案，以允許 SDK 啟用和停用擴音功能：

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

或者，若要手動安裝 SDK，請從此位置下載最新版本：

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

請務必下載附加 `-stages` 的 `aar`。

請求權限

您的應用程式必須請求許可才能存取使用者的攝影機和麥克風。(這不限於 Amazon IVS；任何需要存取攝影機和麥克風的應用程式都必須如此)。

在這裡，我們檢查使用者是否已經授予權限，如果沒有則提出請求：

```
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;
    }
}
```

在這裡，我們取得使用者的回應：

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

發布與訂閱

概念

以下是三個以即時功能為基礎的核心概念：階段、策略和轉譯器。設計目標是盡可能減少打造工作產品所需的用戶端邏輯數量。

階段

Stage 類別是主持人應用程式和 SDK 之間的主要交互點。它代表階段本身，用於加入和離開階段。建立和加入階段需有效且未過期的控制平面權杖字串 (表示為 token)。加入和離開階段並不難。

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

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

stage.leave();
```

Stage 類別也可連接 StageRenderer：

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

策略

Stage.Strategy 介面為主持人應用程式提供了將所需階段狀態傳送至 SDK 的管道。您必須實作以下三項函數：shouldSubscribeToParticipant、shouldPublishFromParticipant、和 stageStreamsToPublishForParticipant。以下將討論所有內容。

訂閱參與者

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

遠端參與者加入階段時，SDK 會向主持人應用程式查詢該參與者所需的訂閱狀態。選項包括 NONE、AUDIO_ONLY 和 AUDIO_VIDEO。傳回此函數的值時，主持人應用程式不需要擔心發布狀態、目前的訂閱狀態或階段連線狀態。若傳回 AUDIO_VIDEO，SDK 會等到遠端參與者發布時才會訂閱，然後在整個程序中透過轉譯器更新主持人應用程式。

以下是實作範例：

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

對於一律希望所有參與者互相看到彼此的主持人應用程式 (例如影片聊天應用程式)，這是此函數的完整實作程序。

您也可以採用更進階的實作方式。在 ParticipantInfo 上使用 userInfo 屬性，以根據伺服器提供的屬性選擇性訂閱參與者：

```
@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;
```

```
 }
}
```

這可以用來建立一個階段，版主可以在不會被看到或聽到自己聲音的情況下監控所有訪客。主持人應用程式可以使用其他商業邏輯，讓版主看到彼此，但仍維持訪客看不到他們的狀態。

發布

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

連線到階段後，SDK 會查詢主持人應用程式，看看特定參與者是否應發布。系統僅會根據提供的字符為具有發布許可的本機參與者調用此函數。

以下是實作範例：

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

這是針對一個使用者總是想發布內容的標準影片聊天應用程式。他們可以靜音和取消靜音其音訊和影片內容，以立即隱藏起來，或看到/聽見內容。(他們也可以使用發布/取消發布，但這種方式速度較慢。建議在需經常變更可見性的使用案例中使用靜音/取消靜音。)

選擇要發布的串流

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

發布時，這會用來決定應發布哪些音訊和影片串流。稍後會在發布媒體串流中進行詳細說明。

更新策略

策略應處於動態狀態：從上述任何函數返回的值可以隨時進行修改。例如，若主持人應用程式在終端使用者按下按鈕前都不想發布，您可以從 shouldPublishFromParticipant 傳回一個變數 (例如 hasUserTappedPublishButton)。當該變數根據終端使用者的互動而變更時，請呼叫

stage.refreshStrategy() 向 SDK 傳送訊號，表示它應查詢策略中的最新值，並僅套用已變更的項目。若 SDK 發現 shouldPublishFromParticipant 值已變更，它便會開始發布程序。若 SDK 查詢後所有函數傳回與之前相同的值，則 refreshStrategy 呼叫將不會對階段進行任何修改。

若 shouldSubscribeToParticipant 傳回的值從 AUDIO_VIDEO 變更為 AUDIO_ONLY，則系統將會針對傳回值已變更的所有參與者移除影片串流 (若之前存有影片串流)。

一般而言，階段會採用策略，以最有效率的方式套用先前與目前策略之間的差異，主持人應用程式不必擔心正確進行管理所需的所有狀態。因此，請將呼叫 stage.refreshStrategy() 視為低成本的操作，因為除非策略發生變化，否則它什麼都不會執行。

轉譯器

StageRenderer 介面會將階段狀態傳送給主持人應用程式。主持人應用程式的 UI 更新通常可以完全由轉譯器提供的事件提供支援。轉譯器會提供以下函數：

```
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);
```

針對大多數這些方法提供了對應的 Stage 和 ParticipantInfo。

轉譯器提供的資訊應該不會對策略的傳回值造成影響。例如，呼叫 onParticipantPublishStateChanged 時，shouldSubscribeToParticipant 的傳回值應該不會變更。若主持人應用程式想要訂閱特定參與者，則無論該參與者的發布狀態為何，它都應傳回所需的訂閱類型。SDK 負責確保根據階段狀態，在正確的時間點執行策略的所需狀態。

您可以將 StageRenderer 連接至階段類別：

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

請注意，當發布參與者觸發 onParticipantJoined，且參與者停止發布或離開階段工作階段時，onParticipantLeft 才會觸發。

發布媒體串流

您可以透過 DeviceDiscovery 找到本地裝置 (例如內建的麥克風和攝影機)。以下是選擇前置攝影機和預設麥克風，然後將其以 LocalStageStreams 傳回並由 SDK 發布的範例：

```
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;
}
```

顯示和移除參與者

訂閱完成後，您會透過轉譯器的 onStreamsAdded 函數收到 StageStream 物件陣列。您可以透過 ImageStageStream 擷取預覽畫面：

```
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);
```

您可以透過 AudioStageStream 擷取音訊層級的統計資料：

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

當參與者停止發布或取消訂閱時，系統會呼叫 onStreamsRemoved 函數，並傳回遭移除的串流。主持人應用程式應將此視為從檢視階層中移除參與者影片串流的信號。

onStreamsRemoved 會在串流可能遭移除的所有情況下調用，其中包括：

- 遠端參與者停止發布。
- 本機裝置取消訂閱，或將訂閱從 AUDIO_VIDEO 變更為 AUDIO_ONLY。
- 遠端參與者離開階段。
- 本機參與者離開階段。

由於 onStreamsRemoved 會在所有情況下調用，因此在遠端或本機離開操作期間，不需要使用自訂商業邏輯從 UI 移除參與者。

靜音和取消靜音媒體串流

LocalStageStream 物件具備控制是否將串流靜音的 setMuted 函數。此函數可以在從 streamsToPublishForParticipant 策略函數傳回之前或之後在串流上呼叫。

重要：如果呼叫 refreshStrategy 後由 streamsToPublishForParticipant 傳回新的 LocalStageStream 物件執行個體，則新串流物件的靜音狀態會套用至階段。建立新 LocalStageStream 執行個體時請務必小心，以確保維持預期的靜音狀態。

監控遠端參與者媒體靜音狀態

當參與者變更其影片或音訊串流的靜音狀態時，會以已變更的串流清單調用轉譯器 onStreamMutedChanged 函數。使用 StageStream 上的 getMuted 方法來據此更新您的 UI。

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

取得 WebRTC 統計資料

若要取得發布串流或訂閱串流的最新 WebRTC 統計資料，請在 StageStream 上使用 requestRTCStats。收集完成後，您將透過可以在上 StageStream 設定的 StageStream.Listener 收到統計資料。

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

取得參與者屬性

如果您在 CreateParticipantToken 端點要求中指定屬性，您可以在 ParticipantInfo 屬性中看到屬性：

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

在背景繼續工作階段

當應用程式進入後台時，建議您停止發布或僅訂閱其他遠程參與者的音訊。若要完成此操作，請更新您的 Strategy 實作以停止發布，並訂閱 AUDIO_ONLY (或 NONE，如適用)。

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

啟用/停用使用 Simulcast 進行分層編碼

發布媒體串流時，SDK 會傳輸高品質和低品質的影片串流，因此即使下行頻寬有限，遠端參與者也可訂閱串流。預設會啟用使用 Simulcast 進行分層編碼。您可以使用 `StageVideoConfiguration.Simulcast` 類別將其停用：

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

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera, config);

// Other Stage implementation code
```

影片組態限制

SDK 不支援使用 `StageVideoConfiguration.setSize(BroadcastConfiguration.Vec2 size)` 強制執行縱向模式或橫向模式。在縱向模式中，較小的空間為寬度；在橫向模式中，較小的空間則為高度。這表示以下兩次 `setSize` 呼叫會對影片組態產生一樣的效果：

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

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

處理網路問題

當本機裝置的網路連線中斷時，SDK 會在內部嘗試重新連線，無需使用者採取任何動作。SDK 在部分情況下會執行失敗，這時就需要使用者採取動作。以下是兩個有關網路連線中斷的主要錯誤：

- 錯誤代碼 1400，訊息：「PeerConnection 因為未知的網路錯誤而遺失」
- 錯誤代碼 1300，訊息：「已用盡重試嘗試次數」

若收到第一種錯誤，但未收到第二種錯誤，SDK 仍會連線至階段，並嘗試自動重新建立連線。保險起見，您可以在不對策略方法的傳回值進行任何更改的情況下呼叫 `refreshStrategy`，以觸發手動重新連線嘗試。

若收到第二種錯誤，則表示 SDK 的重新連線嘗試失敗，且本機裝置已中斷與階段的連線。在此情況下，請嘗試在重新建立網路連線後，呼叫 join 來重新加入階段。

一般而言，若成功加入階段後遇到錯誤，則表示 SDK 並沒有成功重新建立連線。建立新的 Stage 物件，並在網路情況改善時嘗試加入。

使用藍牙麥克風

若要使用藍牙麥克風裝置發布，您必須啟動藍牙 SCO 連線：

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

已知問題和解決方法

- Android 裝置進入睡眠模式後被喚醒時，預覽畫面可能會卡住。

  解決方法： 建立並使用新的 Stage。
- 當參與者以其他參與者正在使用的權杖加入時，第一個連線會中斷連線，且不會顯示具體的錯誤。

  解決方法：無。
- 發布者處於發布狀態但訂閱者收到的發布狀態為 inactive 的情況很少見。

  解決方法：嘗試離開工作階段後再重新加入。若問題仍無法解決，請為發布者建立新權杖。
- 極少數情況下，階段工作階段期間可能會斷斷續續出現音訊失真的問題 (通常是呼叫時間較長時會出現)。

  解決方法：音訊失真的參與者可以離開工作階段後再重新加入，或取消發布其音訊後再重新發布，以便修正此問題。
- 發布至階段時，系統不支援外接麥克風。

  解決方法： 發布至階段時，請勿使用透過 USB 連接的外接麥克風。
- 系統不支援使用 createSystemCaptureSources 發布至螢幕共用的階段。

  解決方法： 使用自訂影像輸入來源和自訂音訊輸入來源手動管理系統擷取。

- 從父項中移除 ImagePreviewView(例如在父項呼叫 removeView( )) 時，系統會立即釋出 ImagePreviewView。將 ImagePreviewView 加至其他父項視圖時，它不會顯示任何畫面。

  解決方法： 使用 getPreview 要求再次預覽。
- 使用作業系統為 Android 12 的 Samsung Galaxy S22/+ 加入階段時，您可能會遭遇 1401 錯誤，且本地裝置可能會無法加入階段，或加入後沒有音訊。

  解決方法： 升級至 Android 13 作業系統。
- 使用作業系統為 Android 13 的 Nokia X20 加入階段時，攝影機可能會無法打開，並出現異常狀況。

  解決方法：無。
- 配備 MediaTek Helio 晶片組的裝置可能無法正確轉譯遠端參與者的視訊。

  解決方法：無。
- 在少數裝置上，裝置作業系統可能會選擇與 SDK 選取的麥克風不同的麥克風。這是因為 Amazon IVS 廣播 SDK 無法控制 VOICE_COMMUNICATION 音訊路由的定義方式，因為它會根據不同的裝置製造商而有所不同。

  解決方法：無。
- 某些安卓視訊編碼器無法設定小於 176x176 的視訊大小。配置較小的大小會導致錯誤並阻止流式傳輸。

  因應措施：請勿將視訊大小設定為小於 176x176。

錯誤處理

嚴重錯誤與非嚴重錯誤

錯誤物件的 BroadcastException 布林值欄位為「is fatal」。

一般而言，嚴重錯誤與 Stages 伺服器的連線有關 (無法建立連線或失去連線且無法復原)。在使用新的權杖或是裝置連線恢復時，應用程式應重新建立階段並重新加入。

非嚴重錯誤通常與發布/訂閱狀態有關，且是由 SDK 處理重試發布/訂閱的作業。

您可以檢查以下屬性：

```
try {
  stage.join(...)
```

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

加入錯誤

權杖格式錯誤

當階段權杖格式不正確時，就會發生此錯誤。

SDK 會從 `stage.join` 呼叫擲出一個 Java 例外狀況，包含 error code = 1000 及 fatal = true。

動作：建立一個有效權杖，然後重試加入。

權杖過期

當階段權杖過期時，就會發生此錯誤。

SDK 會從 `stage.join` 呼叫擲出一個 Java 例外狀況，包含 error code = 1001 及 fatal = true。

動作：建立一個新權杖，然後重試加入。

權杖無效或撤銷

當階段權杖格式正確但遭 Stages 伺服器拒絕時，就會發生此錯誤。此錯誤是透過應用程式提供的階段轉譯器以非同步方式報告。

SDK 會以例外狀況呼叫 `onConnectionStateChanged`，包含 error code = 1026 及 fatal = true。

動作：建立一個有效權杖，然後重試加入。

初始加入時出現網路錯誤

當 SDK 無法聯絡 Stages 伺服器以建立連線時，就會發生此錯誤。此錯誤是透過應用程式提供的階段轉譯器以非同步方式報告。

SDK 會以例外狀況呼叫 `onConnectionStateChanged`，包含 error code = 1300 及 fatal = true。

動作：等待裝置的連線復原，然後重試加入。

已加入時出現網路錯誤

如果裝置的網路連線中斷，SDK 可能會失去與 Stage 伺服器的連線。此錯誤是透過應用程式提供的階段轉譯器以非同步方式報告。

SDK 會以例外狀況呼叫 onConnectionStateChanged，包含 error code = 1300 及 fatal = true。

動作：等待裝置的連線復原，然後重試加入。

發布/訂閱錯誤

初始

錯誤包含以下幾種：

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

這些錯誤是透過應用程式提供的階段轉譯器以非同步方式報告。

SDK 會重試作業，但次數有限。在重試期間，發布/訂閱狀態為 ATTEMPTING_PUBLISH / ATTEMPTING_SUBSCRIBE。如果重試成功，狀態會變更為 PUBLISHED / SUBSCRIBED。

SDK 呼叫 onError 包含相關的錯誤代碼，且 fatal = false。

動作：SDK 會自動重試，因此不需執行任何動作。或者，應用程式可以重新整理策略以強制執行更多次重試。

建立後失敗

發布或訂閱可能會在建立後失敗，這很可能是因為網路錯誤所致。「對等連線因網路錯誤而中斷」訊息的錯誤代碼是 1400。

此錯誤是透過應用程式提供的階段轉譯器以非同步方式報告。

SDK 會重試發布/訂閱作業。在重試期間，發布/訂閱狀態為 ATTEMPTING_PUBLISH / ATTEMPTING_SUBSCRIBE。如果重試成功，狀態會變更為 PUBLISHED / SUBSCRIBED。

SDK 呼叫 onError 包含 error code = 1400 及 fatal = false。

動作：SDK 會自動重試，因此不需執行任何動作。或者，應用程式可以重新整理策略以強制執行更多次重試。在網路完全無法連線的情況下，與 Stages 的連線可能也會失敗。

IVS 廣播 SDK：iOS 指南 (即時串流)

IVS 即時串流 iOS 廣播 SDK 讓參與者能夠在 iOS 上傳送和接收影片。

`AmazonIVSBroadcast` 模組會實作本文件中所述的界面。支援以下操作：

- 加入階段
- 將媒體發布給階段中的其他參與者
- 訂閱階段中其他參與者的媒體
- 管理和監控發布到階段的影片和音訊
- 取得每個對等連線的 WebRTC 統計資料
- IVS 低延遲串流 iOS 廣播 SDK 的所有操作

iOS 廣播軟體開發套件的最新版本：1.14.1 (版本說明)

參考文件：如需 Amazon IVS iOS 廣播開發套件中最重要方法的相關資訊，請參閱參考文件，網址為 https://aws.github.io/ amazon-ivs-broadcast-docs /1.14.1/ios/。

示例代碼：請參閱以下位置的 iOS 示例存儲庫 GitHub：https://github.com/aws-samples/ amazon-ivs-broadcast-ios-示例。

平台需求：iOS 14 或更高版本。

開始

安裝程式庫

我們建議您透過整合廣播 SDK CocoaPods。(或者，您可以手動將架構新增到您的專案中)。

建議：整合廣播 SDK (CocoaPods)

即時功能做為 iOS 低延遲串流廣播 SDK 的子規格發布。這樣客戶就可以根據自己的功能需求選擇納入或排除功能。納入功能可提升套件大小。

發行版本是透過 CocoaPods 名稱發佈的`AmazonIVSBroadcast`。將此相依性新增到您的 Podfile：

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

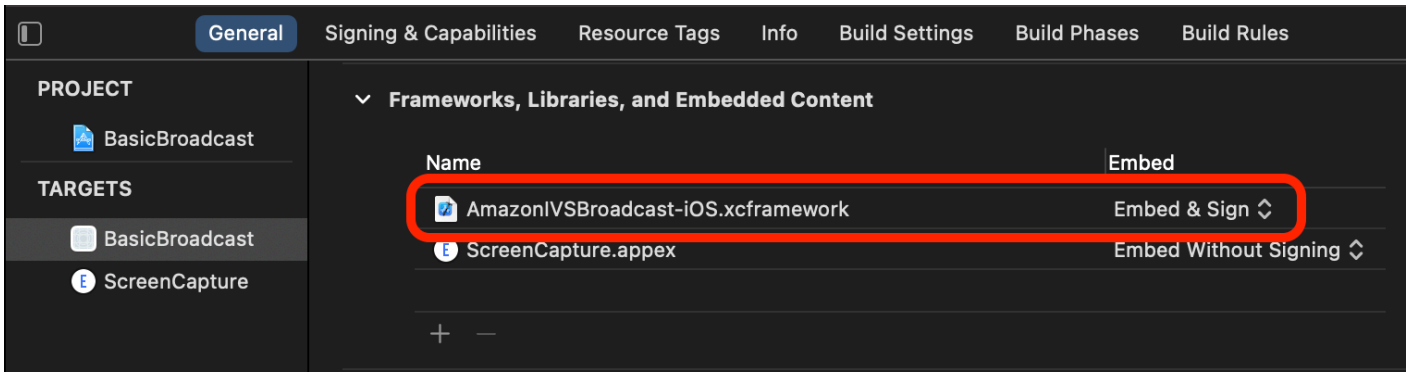
執行 `pod install`，將可在您的 `.xcworkspace` 中使用開發套件。

重要：IVS 即時串流廣播 SDK (即具有階段子規格) 包括 IVS 低延遲串流廣播 SDK 的所有功能。您無法將這兩個 SDK 整合進同一個專案。如果您將階段子規格添加 CocoaPods 到項目中，請確保刪除 Podfile 中包含的任何其他行。`AmazonIVSBroadcast`例如，請不要在 Podfile 中同時含有以下這兩行：

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

替代方法：手動安裝架構

1. 請從以下位置下載最新版本：https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip。
2. 解壓縮封存檔的內容。`AmazonIVSBroadcast.xcframework` 包含用於裝置和模擬器的開發套件。
3. 內嵌 `AmazonIVSBroadcast.xcframework`，方法是將其拖曳至您的應用程式目標的一般索引標籤的架構、程式庫和內嵌內容部分中。



請求權限

您的應用程式必須請求許可才能存取使用者的攝影機和麥克風。(這不限於 Amazon IVS；任何需要存取攝影機和麥克風的應用程式都必須如此)。

在這裡，我們檢查使用者是否已經授予許可，如果沒有則提出請求：

```
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.
}
```

如果您想要存取攝影機和麥克風，必須分別對 .video 和 .audio 媒體類型執行此動作。

此外，您必須將 NSCameraUsageDescription 和 NSMicrophoneUsageDescription 的項目新增至您的 Info.plist。否則，您的應用程式將在嘗試請求許可時當機。

停用應用程式閒置計時器

此為選用操作，但建議您採用。這可以防止您的裝置在使用廣播開發套件時進入休眠狀態，導致廣播中斷。

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

發布與訂閱

概念

以下是三個以即時功能為基礎的核心概念：階段、策略和轉譯器。設計目標是盡可能減少打造工作產品所需的用戶端邏輯數量。

階段

IVSStage 類別是主持人應用程式和 SDK 之間的主要交互點。該類別代表階段本身，用於加入和離開階段。建立或加入階段需有效且未過期的控制平面字符字串 (表示為 token)。加入和離開階段並不難。

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

```
try stage.join()

stage.leave()
```

IVSStage 類別也可連接 IVSStageRenderer 和 IVSErrorDelegate：

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

策略

IVSStageStrategy 協定為主持人應用程式提供了將所需階段狀態傳送至 SDK 的管道。您必須實作以下三項函數：shouldSubscribeToParticipant、shouldPublishParticipant、和 streamsToPublishForParticipant。以下將討論所有內容。

訂閱參與者

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

遠端參與者加入階段時，SDK 會向主持人應用程式查詢該參與者所需的訂閱狀態。選項包括 .none、.audioOnly 和 .audioVideo。傳回此函數的值時，主持人應用程式不需要擔心發布狀態、目前的訂閱狀態或階段連線狀態。若傳回 .audioVideo，SDK 會等到遠端參與者發布時才會訂閱，然後在整個程序中透過轉譯器更新主持人應用程式。

以下是實作範例：

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

對於一律希望所有參與者互相看到彼此的主持人應用程式 (例如影片聊天應用程式)，這是此函數的完整實作程序。

您也可以採用更進階的實作方式。在 IVSParticipantInfo 上使用 attributes 屬性，以根據伺服器提供的屬性選擇性訂閱參與者：

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

```
    switch participant.attributes["role"] {
    case "moderator": return .none
    case "guest": return .audioVideo
    default: return .none
    }
}
```

這可以用來建立一個階段，版主可以在不會被看到或聽到自己聲音的情況下監控所有訪客。主持人應用程式可以使用其他商業邏輯，讓版主看到彼此，但仍維持訪客看不到他們的狀態。

發布

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

連線到階段後，SDK 會查詢主持人應用程式，看看特定參與者是否應發布。系統僅會根據提供的字符為具有發布許可的本機參與者調用此函數。

以下是實作範例：

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

這是針對一個使用者總是想發布內容的標準影片聊天應用程式。他們可以靜音和取消靜音其音訊和影片內容，以立即隱藏起來，或看到/聽見內容。(他們也可以使用發布/取消發布，但這種方式速度較慢。建議在需經常變更可見性的使用案例中使用靜音/取消靜音。)

選擇要發布的串流

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

發布時，這會用來決定應發布哪些音訊和影片串流。稍後會在發布媒體串流中進行詳細說明。

更新策略

策略應處於動態狀態：從上述任何函數返回的值可以隨時進行修改。例如，若主持人應用程式在終端使用者按下按鈕前都不想發布，您可以從 shouldPublishParticipant 傳回一個變數 (例如 hasUserTappedPublishButton)。當該變數根據終端使用者的互動而變更時，請呼叫

stage.refreshStrategy() 向 SDK 傳送訊號，表示它應查詢策略中的最新值，並僅套用已變更的項目。若 SDK 發現 shouldPublishParticipant 值已變更，它便會開始發布程序。若 SDK 查詢後所有函數傳回與之前相同的值，則 refreshStrategy 呼叫將不會對階段進行任何修改。

若 shouldSubscribeToParticipant 傳回的值從 .audioVideo 變更為 .audioOnly，則系統將會針對傳回值已變更的所有參與者移除影片串流 (若之前存有影片串流)。

一般而言，階段會採用策略，以最有效率的方式套用先前與目前策略之間的差異，主持人應用程式不必擔心正確進行管理所需的所有狀態。因此，請將呼叫 stage.refreshStrategy() 視為低成本的操作，因為除非策略發生變化，否則它什麼都不會執行。

轉譯器

IVSStageRenderer 協定會將階段狀態傳送給主持人應用程式。主機應用程式的 UI 更新通常可以完全由轉譯器提供的事件提供支援。轉譯器會提供以下函數：

```
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?)
```

轉譯器提供的資訊應該不會對策略的傳回值造成影響。例如，呼叫 participant:didChangePublishState 時，shouldSubscribeToParticipant 的傳回值應該不會變更。若主持人應用程式想要訂閱特定參與者，則無論該參與者的發布狀態為何，它都應傳回所需的訂閱類型。SDK 負責確保根據階段狀態，在正確的時間點執行策略的所需狀態。

請注意，當發布參與者觸發 participantDidJoin，且參與者停止發布或離開階段工作階段時，participantDidLeave 才會觸發。

發布媒體串流

您可以透過 IVSDeviceDiscovery 找到本地裝置 (例如內建的麥克風和攝影機)。以下是選擇前置攝影機和預設麥克風，然後將其以 IVSLocalStageStreams 返回並由 SDK 發布的範例：

```
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]
}
```

顯示和移除參與者

訂閱完成後，您會透過轉譯器的 didAddStreams 函數收到 IVSStageStream 物件陣列。若要預覽或接收有關此參與者的音訊層級統計資料，您可以從串流中存取基礎 IVSDevice 物件：

```
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 */
    })
}
```

當參與者停止發布或取消訂閱時，系統會呼叫 didRemoveStreams 函數，並傳回遭移除的串流。主持人應用程式應將此視為從檢視階層中移除參與者影片串流的信號。

didRemoveStreams 會在串流可能遭移除的所有情況下調用，其中包括：

- 遠端參與者停止發布。
- 本機裝置取消訂閱，或將訂閱從 .audioVideo 變更為 .audioOnly。
- 遠端參與者離開階段。
- 本機參與者離開階段。

由於 didRemoveStreams 會在所有情況下調用，因此在遠端或本機離開操作期間，不需要使用自訂商業邏輯從 UI 移除參與者。

靜音和取消靜音媒體串流

IVSLocalStageStream 物件具備控制是否將串流靜音的 setMuted 函數。此函數可以在從 streamsToPublishForParticipant 策略函數傳回之前或之後在串流上呼叫。

重要：如果呼叫 refreshStrategy 後由 streamsToPublishForParticipant 傳回新的 IVSLocalStageStream 物件執行個體，則新串流物件的靜音狀態會套用至階段。建立新 IVSLocalStageStream 執行個體時請務必小心，以確保維持預期的靜音狀態。

監控遠端參與者媒體靜音狀態

當參與者變更其影片或音訊串流的靜音狀態時，會以已變更的串流陣列調用轉譯器 didChangeMutedStreams 函數。使用 IVSStageStream 上的 isMuted 屬性來據此更新您的 UI：

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

建立階段組態

若要自訂階段影片組態的值，請使用 IVSLocalStageStreamVideoConfiguration：

```
let config = IVSLocalStageStreamVideoConfiguration()
```

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

取得 WebRTC 統計資料

若要取得發布串流或訂閱串流的最新 WebRTC 統計資料，請在 IVSStageStream 上使用 requestRTCStats。收集完成後，您將透過可以在上 IVSStageStream 設定的 IVSStageStreamDelegate 收到統計資料。若要持續收集 WebRTC 統計資料，請透過 Timer 呼叫此函數。

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

取得參與者屬性

如果您在 CreateParticipantToken 端點要求中指定屬性，您可以在 IVSParticipantInfo 屬性中看到屬性：

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

在背景繼續工作階段

當應用程式進入背景時，您可以在聽到遠端音訊的同時繼續待在階段，不過無法繼續傳送自己的影像和音訊。您必須更新您的 IVSStrategy 實作以停止發布，並訂閱 .audioOnly (或 .none，如適用)。

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

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

然後呼叫 `stage.refreshStrategy()`。

啟用/停用使用 Simulcast 進行分層編碼

發布媒體串流時，SDK 會傳輸高品質和低品質的影片串流，因此即使下行頻寬有限，遠端參與者也可訂閱串流。預設會啟用使用 Simulcast 進行分層編碼。您可以使用 `IVSSimulcastConfiguration` 將其停用:

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

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

// Other Stage implementation code
```

將階段廣播到 IVS 頻道

若要廣播階段，請建立一個獨立 `IVSBroadcastSession`，然後按照使用 SDK 進行廣播的一般指示操作 (如上所述)。`IVSStageStream` 的 `device` 屬性會是 `IVSImageDevice` 或 `IVSAudioDevice`，如上述程式碼片段所示；這些屬性可以連線至 `IVSBroadcastSession.mixer`，以可自訂的版面配置廣播整個階段。

或者，您可以複合階段並將其廣播到 IVS 低延遲通道，以吸引更多受眾。請參閱《IVS 低延遲串流使用者指南》中的在 Amazon IVS 串流上啟用多位主持人。

iOS 如何選擇攝影機解析度和影格速率

由廣播 SDK 管理的攝影機可最佳化其解析度和影格速率 (或 FPS)frames-per-second，以將熱量產生和能源消耗降到最低。本節說明如何選取解析度和影格速率，以協助主持人應用程式針對其使用案例進行優化。

當利用 `IVSCamera` 建立 `IVSLocalStageStream` 時，會針對 `IVSLocalStageStreamVideoConfiguration.targetFramerate` 的影格速率

和 IVSLocalStageStreamVideoConfiguration.size 的解析度優化攝影機。呼叫 IVSLocalStageStream.setConfiguration 會以較新的值更新攝影機。

攝影機預覽

如果您建立 IVSCamera 的預覽但未將其連接至 IVSBroadcastSession 或 IVSStage，則預設為 1080p 解析度，影格速率為 60 fps。

廣播階段

使用 IVSBroadcastSession 來廣播 IVSStage 時，SDK 會嘗試使用符合兩個工作階段標準的解析度和影格速率來優化攝影機。

例如，如果廣播組態設定為 15 FPS 的影格速率且解析度為 1080p，而階段的影格速率為 30 FPS 且解析度為 720p，則 SDK 會選擇影格速率為 30 FPS 且解析度為 1080p 的攝影機組態。IVSBroadcastSession 將丟棄攝影機中的所有其他影格，且 IVSStage 會將 1080p 的圖像縮小為 720p。

如果主持人應用程式計畫同時將 IVSBroadcastSession 和 IVSStage 與攝影機搭配使用，建議各自組態的 targetFramerate 和 size 屬性應相匹配。不匹配可能會導致攝影機在擷取影片時重新自行設定，而這將導致影片樣本傳遞的短暫延遲。

如果具有相同的值不符合主持人應用程式的使用案例，則先建立的較高品質攝影機將防止攝影機在新增品質較低的工作階段時重新自行設定。例如，如果您以 1080p 和 30 FPS 進行廣播，然後再加入設定為 720p 和 30 FPS 的階段，則攝影機將不會自行重新設定，而且影片也會繼續而不會中斷。這是因為 720p 小於或等於 1080p，且 30 FPS 小於或等於 30 FPS。

任意影格速率、解析度和長寬比

大多數攝影機硬體能與常見格式完全匹配，例如 30 FPS 時為 720p 或 60 FPS 時為 1080p。不過，無法與所有格式完全匹配。廣播 SDK 根據以下規則 (按優先順序) 選擇攝影機組態：

1. 解析度的寬度和高度大於或等於所需的解析度，但在此限制中，寬度和高度越小越好。
2. 影格速率大於或等於所需的影格速率，但在此限制範圍內，影格速率越低越好。
3. 長寬比與所需的長寬比相匹配。
4. 如果有多種匹配格式，則使用具有最大視野的格式。

以下是兩個範例：

- 主持人應用程式正在嘗試以 120 FPS 的速率以 4k 進行廣播。選取的攝影機僅支援 60 FPS 時為 4K，或 120 FPS 時為 1080p。選取的格式將會是 60 FPS 時為 4k，因為解析度規則的優先順序高於影格速率規則。
- 請求不規則的解析度，即 1910x1070。攝影機將使用 1920x1080。請注意：選擇 1921x1080 之類的解析度將導致攝影機縱向擴展到下一個可用的解析度 (例如 2592x1944)，這會造成 CPU 和記憶體頻寬的損失。

那 Android 呢？

Android 不會像 iOS 那樣立即調整其解析度或影格速率，因此這不會影響 Android 廣播 SDK。

已知問題和解決方法

- 變更藍牙音訊路由可能無法預測。如果您在工作階段中連接新裝置，iOS 可能會自動變更輸入路由。此外，您無法在同一時間連接的多個藍牙耳機之間進行選擇。這會出現在一般廣播和階段工作階段中。

  解決方法： 如果您打算使用藍牙耳機，請在開始廣播或階段之前先連接耳機，並在整個工作階段保持連線狀態。
- 使用 iPhone 14、iPhone 14 Plus、iPhone 14 Pro 或 iPhone 14 Pro Max 的參與者可能會導致其他參與者的音訊產生回音問題。

  解決方法：使用受影響裝置的參與者可以使用耳機來防止其他參與者出現回音問題。
- 當參與者以其他參與者正在使用的權杖加入時，第一個連線會中斷連線，且不會顯示具體的錯誤。

  解決方法：無。
- 發布者處於發布狀態但訂閱者收到的發布狀態為 `inactive` 的情況很少見。

  解決方法：嘗試離開工作階段後再重新加入。若問題仍無法解決，請為發布者建立新權杖。
- 當參與者正在發布或訂閱時，即使網路穩定，也可能會收到代碼 1400 的錯誤，表示由於網路問題導致連線中斷。

  解決方法：嘗試重新發布/重新訂閱。
- 極少數情況下，階段工作階段期間可能會斷斷續續出現音訊失真的問題 (通常是呼叫時間較長時會出現)。

  解決方法：音訊失真的參與者可以離開工作階段後再重新加入，或取消發布其音訊後再重新發布，以便修正此問題。

錯誤處理

嚴重錯誤與非嚴重錯誤

錯誤物件的布林值為「is fatal」。這是 `IVSBroadcastErrorIsFatalKey` 底下的字典項目，包含一個布林值。

一般而言，嚴重錯誤與 Stages 伺服器的連線有關 (無法建立連線或失去連線且無法復原)。在使用新的權杖或是裝置連線恢復時，應用程式應重新建立階段並重新加入。

非嚴重錯誤通常與發布/訂閱狀態有關，且是由 SDK 處理重試發布/訂閱的作業。

您可以檢查以下屬性：

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

加入錯誤

權杖格式錯誤

當階段權杖格式不正確時，就會發生此錯誤。

該 SDK 拋出一個斯威夫特異常，錯誤代碼 = 1000 和 IVS BroadcastErrorIsFatalKey = 是。

動作：建立一個有效權杖，然後重試加入。

權杖過期

當階段權杖過期時，就會發生此錯誤。

該 SDK 拋出一個斯威夫特異常錯誤代碼 = 1001 和 IVS BroadcastErrorIsFatalKey = 是。

動作：建立一個新權杖，然後重試加入。

權杖無效或撤銷

當階段權杖格式正確但遭 Stages 伺服器拒絕時，就會發生此錯誤。此錯誤是透過應用程式提供的階段轉譯器以非同步方式報告。

SDK 呼叫時`stage(didChange connectionState, withError error)`，錯誤碼為 1026 且 IVS BroadcastErrorIsFatalKey = 是。

動作：建立一個有效權杖，然後重試加入。

初始加入時出現網路錯誤

當 SDK 無法聯絡 Stages 伺服器以建立連線時，就會發生此錯誤。此錯誤是透過應用程式提供的階段轉譯器以非同步方式報告。

SDK 呼叫時`stage(didChange connectionState, withError error)`，錯誤碼為 1300 且 IVS BroadcastErrorIsFatalKey = 是。

動作：等待裝置的連線復原，然後重試加入。

已加入時出現網路錯誤

如果裝置的網路連線中斷，SDK 可能會失去與 Stage 伺服器的連線。此錯誤是透過應用程式提供的階段轉譯器以非同步方式報告。

SDK 呼叫時`stage(didChange connectionState, withError error)`，錯誤碼為 1300 且 IVS BroadcastErrorIsFatalKey 值 = 是。

動作：等待裝置的連線復原，然後重試加入。

發布/訂閱錯誤

初始

錯誤包含以下幾種：

- MultihostSessionOfferCreationFailPublish
- MultihostSessionOfferCreationFailSubscribe
- MultihostSessionNoIceCandidates
- MultihostSessionStageAtCapacity (1024)
- SignallingSessionCannotRead
- SignallingSessionCannotSend
- SignallingSessionBadResponse

這些錯誤是透過應用程式提供的階段轉譯器以非同步方式報告。

SDK 會重試作業，但次數有限。在重試期間，發布/訂閱狀態為 `ATTEMPTING_PUBLISH` / `ATTEMPTING_SUBSCRIBE`。如果重試成功，狀態會變更為 `PUBLISHED` / `SUBSCRIBED`。

SDK 會IVSErrorSourceDelegate:didEmitError使用相關的錯誤碼呼叫，而 IVS BroadcastErrorIsFatalKey = 否。

動作：SDK 會自動重試，因此不需執行任何動作。或者，應用程式可以重新整理策略以強制執行更多次重試。

建立後失敗

發布或訂閱可能會在建立後失敗，這很可能是因為網路錯誤所致。「對等連線因網路錯誤而中斷」訊息的錯誤代碼是 1400。

此錯誤是透過應用程式提供的階段轉譯器以非同步方式報告。

SDK 會重試發布/訂閱作業。在重試期間，發布/訂閱狀態為 ATTEMPTING_PUBLISH / ATTEMPTING_SUBSCRIBE。如果重試成功，狀態會變更為 PUBLISHED / SUBSCRIBED。

SDK 呼叫didEmitError時發生錯誤代碼為 1400 且 IVS BroadcastErrorIsFatalKey = 否。

動作：SDK 會自動重試，因此不需執行任何動作。或者，應用程式可以重新整理策略以強制執行更多次重試。在網路完全無法連線的情況下，與 Stages 的連線可能也會失敗。

IVS 廣播 SDK：自訂影像來源 (即時串流)

自訂圖像輸入來源讓應用程式能將自己的圖像輸入提供給廣播開發套件，而不是僅限於預設攝影機。自訂圖像來源可以是簡單的半透明浮水印或靜態的「馬上回來」場景，也可以是允許應用程式執行額外的自訂處理，像是在相機上加上美顏濾鏡。

當您使用自訂圖像輸入來源對相機進行自訂控制時 (例如，使用需要相機存取權的美顏濾鏡程式庫)，就不再由廣播開發套件負責管理相機。而是由應用程式負責正確處理相機的生命週期。請參閱官方平台文件，以了解您的應用程式應該如何管理相機。

Android

建立 DeviceDiscovery 工作階段後，建立圖像輸入來源：

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

此方法會傳回 CustomImageSource，這是一個由標準 Android Surface 支持的圖像來源。SurfaceSource 可以調整大小和旋轉的子類別。您還可以建立 ImagePreviewView 以顯示其內容的預覽。

若要檢索底層 Surface：

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

此 Surface 可以作為圖像製作工具 (像是 Camera2、OpenGL ES 和其他程式庫) 的輸出緩衝。最簡單的使用案例是將靜態點陣圖或顏色直接繪製到 Surface 的 Canvas 中。但是，許多程式庫 (像是美顏濾鏡程式庫) 都有提供一種方法，讓應用程式能指定外部 Surface 進行渲染。你可以使用這樣的方法來將此 Surface 傳遞到濾鏡程式庫，這允許程式庫輸出處理過的影格，以便廣播工作階段進行串流。

此 CustomImageSource 可以包裝在 LocalStageStream 中，並由 StageStrategy 傳回以發布到 Stage。

iOS

建立 DeviceDiscovery 工作階段後，建立圖像輸入來源：

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

這個方法會傳回 IVSCustomImageSource，這是一個允許應用程式手動提交 CMSampleBuffers 的圖像來源。有關支援的像素格式，請參閱 iOS 廣播開發套件參考文件；最新版本的連結位於 Amazon IVS 版本備註中，可以取得最新的廣播開發套件版本。

提交到自訂來源的範例將會串流至舞台：

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

針對串流影片，請在回呼中使用此方法。例如，如果您使用的是相機，則每次從 AVCaptureSession 收到新範本緩衝時，應用程式可以將範本緩衝轉發到自訂圖像來源。如果需要，應用程式可以在將樣本提交給自訂圖像來源之前執行進一步處理 (像是美顏濾鏡)。

該 IVSCustomImageSource 可以包裝在 IVSLocalStageStream 中，並由 IVSStageStrategy 傳回以發布到 Stage。

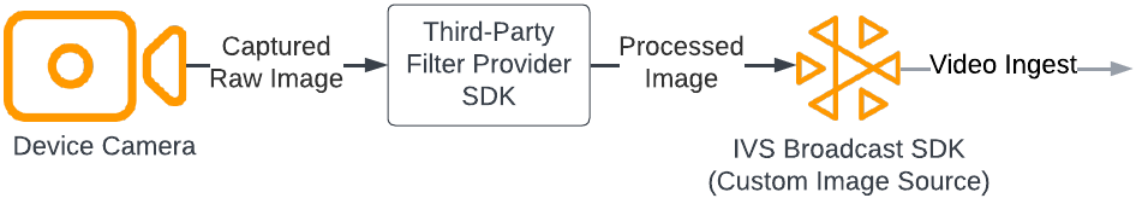
IVS 廣播 SDK：第三方攝影機濾鏡 (即時串流)

本指南假設您已熟悉自訂影像來源，並且已將 IVS 即時串流廣播 SDK 整合到您的應用程式中。

攝影機濾鏡可讓即時串流創作者增強或改變臉部或背景外觀。這可能會增加觀眾參與度，吸引觀眾並增強即時串流體驗。

整合第三方攝影機濾鏡

藉由將濾鏡 SDK 的輸出提供給自訂影像輸入來源，您可以整合第三方攝影機濾鏡 SDK 與 IVS 廣播 SDK。自訂影像輸入來源讓應用程式能將自己的影像輸入提供給廣播 SDK。第三方濾鏡提供者的 SDK 可能會管理攝影機的生命週期，以處理來自攝影機的影像、套用濾鏡效果，並輸出可傳遞至自訂影像來源的格式。



請參閱第三方濾鏡提供者的說明文件，瞭解將套用了濾鏡效果的攝影機影格轉換為可傳遞至自訂影像輸入來源的格式的內建方法。此程序會因為使用的 IVS 廣播 SDK 版本而有所不同：

- Web：濾鏡提供者必須能夠將其輸出轉譯到畫布元素。然後，captureStream 方法可以用來回傳畫布內容的 MediaStream。接著，MediaStream 可以轉換為 LocalStageStream 的執行個體，並發布到舞台。
- Android：濾鏡提供者的 SDK 可以將影格轉譯到 IVS 廣播 SDK 所提供的 Android `Surface`，也可以將影格轉換為點陣圖。如果使用點陣圖，則可以透過解鎖並寫入畫布，將其轉譯到自訂影像來源所提供的基礎 `Surface`。
- iOS：第三方濾鏡提供者的 SDK 必須提供套用了濾鏡效果 `CMSampleBuffer` 的攝影機影格。如需有關如何在處理攝影機影像後取得 `CMSampleBuffer` 作為最終輸出的資訊，請參閱第三方濾鏡廠商 SDK 的文件。

BytePlus

Android

安裝和設定 BytePlus 效果 SDK

有關如何安裝、初始化和設定 BytePlus 效果 SDK 的詳細資訊，請參閱 BytePlus Android Access Guide。

設定自訂影像來源

初始化 SDK 後，將已處理且套用了濾鏡效果的攝影機影格提供給自訂影像輸入源。若要這麼做，請建立 `DeviceDiscovery` 物件的執行個體並建立自訂影像來源。請注意，當您使用自訂影像輸入來源對

攝影機進行自訂控制時，就不再由廣播 SDK 負責管理攝影機。而是由應用程式負責正確處理攝影機的生命週期。

Java

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

將輸出轉換為點陣圖並提供給自訂影像輸入來源

要讓從 BytePlus 效果 SDK 套用濾鏡效果的攝影機影格直接轉至 IVS 廣播 SDK，請將 BytePlus 效果 SDK 的紋理輸出轉換為點陣圖。處理影像時，SDK 會叫用 `onDrawFrame()` 方法。`onDrawFrame()` 方法是 Android GLSurfaceView.Renderer 介面的公用方法。在 BytePlus 提供的 Android 範例應用程式中，此方法會受到每個攝影機影格的呼叫，繼而輸出紋理。同時，您可以使用邏輯來補充 `onDrawFrame()` 方法，將此紋理轉換為點陣圖並將其提供給自訂影像輸入來源。如下列程式碼範例所示，請使用 BytePlus SDK 提供的 `transferTextureToBitmap` 方法來執行此轉換。這個方法由來自 BytePlus 效果 SDK 的 com.bytedance.labcv.core.util.ImageUtil 程式庫提供，如下列程式碼範例所示。然後藉由將產生的點陣圖寫入至 Surface 的 Canvas，將其轉譯到 `CustomImageSource` 的基礎 Android `Surface`。對 `onDrawFrame()` 的許多成功調用會帶來一系列點陣圖，並會在合併時建立影片串流。

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

有關如何整合 DeepAR SDK 與 Android IVS 廣播 SDK 的詳細訊息，請參閱 Android Integration Guide from DeepAR。

iOS

有關如何整合 DeepAR SDK 與 iOS IVS 廣播 SDK 的詳細訊息，請參閱 iOS Integration Guide from DeepAR。

Snap

Web

本節假設您已熟悉使用 Web 廣播 SDK 發布和訂閱影片。

若要整合 Snap 的攝影機套件 SDK 與 IVS 即時串流 Web 廣播 SDK，您需要：

1. 安裝攝影機套件 SDK 和 Webpack。(我們的範例使用 Webpack 作為打包工具，但您可以自行選擇任何打包工具。)
2. 建立 `index.html`。
3. 新增設定元素。
4. 顯示和設定參與者。
5. 顯示連接的攝影機和麥克風。
6. 建立攝影機套件工作階段。
7. 擷取並套用鏡頭。
8. 將攝影機套件工作階段的輸出轉譯至畫布。
9. 為攝影機套件提供用於轉譯和發布 `LocalStageStream` 的媒體來源。
10. 建立一個 Webpack 組態檔。

下文將介紹上述每個步驟。

安裝攝影機套件 SDK 和 Webpack

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

建立 index.html

接下來，建立 HTML 樣板並將 Web 廣播 SDK 匯入為指令碼標籤。在下列程式碼中，請務必用您的廣播 SDK 版本取代 <SDK version>。

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>
```

新增設定元素

建立 HTML 來選取攝影機和麥克風並指定參與者權杖：

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>
```

在其下方新增額外的 HTML 來顯示來自本機和遠端參與者的攝影機供稿：

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>
```

載入額外邏輯，包括用於設定攝影機和已綁定 JavaScript 檔案的輔助方法。(在本節的稍後部分，您要建立這些 JavaScript 檔案並將它們綁定到單一檔案中，以便將攝影機套件匯入為模組。綁定的 JavaScript 檔案將包含設定攝影機套件、套用鏡頭和將套用了鏡頭的攝影機供稿發布到舞台的邏輯。)

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>
```

顯示和設定參與者

接下來建立 `helpers.js`，其中包含您會用來顯示和設定參與者的輔助方法：

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;
}
```

顯示連接的攝影機和麥克風

接下來建立 `media-devices.js`，其中包含用於顯示連接到裝置的攝影機和麥克風的輔助方法：

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,
    },
  });
```

```
}
```

建立攝影機套件工作階段

建立 `stages.js`，其中包含將鏡頭套用至攝影機供稿並將供稿發布至舞台的邏輯。在本檔案的第一部分，我們匯入廣播 SDK 和攝影機套件 Web SDK，並初始化我們將在每個 SDK 中使用的變數。我們在引導攝影機套件 Web SDK 後透過呼叫 `createSession` 建立起攝影機套件工作階段。請注意，畫布元素物件會被傳遞給工作階段；這將告知攝影機套件轉譯至該畫布。

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 });
```

擷取並套用鏡頭

要擷取自己的鏡頭，請插入可以在 Camera Kit Developer Portal 中找到的鏡頭組 ID。在本範例中，我們透過套用回傳的鏡頭陣列中的第一個鏡頭來簡化這一步。

JavaScript

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

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

將攝影機套件工作階段的輸出轉譯到畫布

使用 captureStream 方法回傳畫布內容中的 MediaStream。畫布將包含套用了鏡頭的攝影機供稿的影片串流。此外，新增了用於攝影機和麥克風靜音按鈕的事件接聽程式，以及用於加入和離開舞台的事件接聽程式。在用於加入舞台的事件接聽程式中，我們從畫布傳遞攝影機套件工作階段和 MediaStream，以便將其發布到舞台。

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();
  });
};
```

為攝影機套件提供用於轉譯的媒體來源並發布 LocalStageStream

若要發布套用了鏡頭的影片串流，請建立名為 `setCameraKitSource` 的函數來傳遞稍早從畫布擷取的 `MediaStream`。來自畫布的 `MediaStream` 目前沒有作用，因為我們還未納入本地攝影機供稿。我們可以透過呼叫 `getCamera` 輔助方法並將其分配給 `localCamera` 來合併本地攝影機供稿。然後，我們可以將本地攝影機供稿 (透過 `localCamera`) 和工作階段物件傳遞給 `setCameraKitSource`。`setCameraKitSource` 函數能透過呼叫 `createMediaStreamSource` 將本地攝影機供稿轉換為 CameraKit 媒體來源。接著將 `CameraKit` 媒體來源轉換成前置攝影機的鏡像。然後，鏡頭效果被套用到媒體來源，並通過呼叫 `session.play()` 轉譯到輸出畫布。

此時，鏡頭已套用到擷取自畫布的 `MediaStream`，接著可以繼續將其發布到舞台。使用來自的 `MediaStream` 影片軌道建立 `LocalStageStream`，即可實現此目的。然後，`LocalStageStream` 的執行個體可以傳入到要發布的 `StageStrategy`。

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;
    },
  };
```

下面的其餘代碼用於建立和管理我們的舞台：

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();
```

建立一個 Webpack 組態檔

建立 `webpack.config.js` 並新增以下程式碼。這將上面的邏輯綁定在一起，以便您可以透過 import 陳述式來使用攝影機套件。

JavaScript

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

最後，按照 Webpack 組態檔的定義執行 `npm run build` 來綁定自己的 JavaScript。然後，您可以從 Web 伺服器提供 HTML 和 JavaScript。例如，您可以使用 Python 的 HTTP 伺服器並打開 localhost:8000 來查看結果：

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

Android

若要整合 Snap 的攝影機套件 SDK 與 IVS Android 廣播 SDK，您必須安裝攝影機套件 SDK、初始化攝影機套件工作階段、套用鏡頭，然後將攝影機套件工作階段的輸出提供給自訂影像輸入來源。

要安裝攝影機套件 SDK，請將以下內容新增到模組的 `build.gradle` 檔案中。將 `$cameraKitVersion` 替換為攝影機套件 SDK 的最新版本。

Java

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

初始化並取得 `cameraKitSession`。攝影機套件還為 Android 的 CameraX API 提供了一個方便的包裝函式，讓您無需編寫複雜的邏輯即可共用 CameraX 與攝影機套件。您可以使用 `CameraXImageProcessorSource` 物件作為r ImageProcessor 的 Source，讓自己啟動攝影機預覽串流影格。

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();
    }
```

擷取並套用鏡頭

您可以在 Camera Kit Developer Portal 的輪播中設定並訂購鏡頭：

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

若要廣播，請將已處理的影格傳送至自訂影像來源的基礎 Surface。使用 DeviceDiscovery 物件並建立 CustomImageSource 來回傳 SurfaceSource。然後，您可以將 CameraKit 工作階段的輸出轉譯至由 SurfaceSource 提供的基礎 Surface。

Java

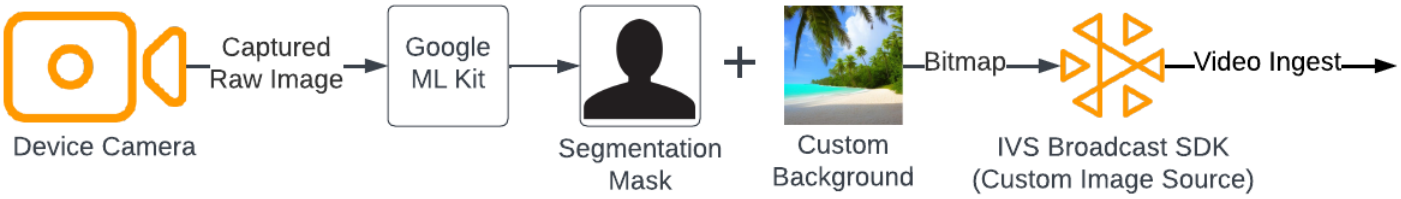
```
val publishStreams = ArrayList<LocalStageStream>()

val deviceDiscovery = DeviceDiscovery(applicationContext)
val customSource =
 deviceDiscovery.createImageInputSource(BroadcastConfiguration.Vec2(720f, 1280f))

cameraKitSession.processor.connectOutput(outputFrom(customSource.inputSurface))
val customStream = ImageLocalStageStream(customSource)

// After rendering the output from a Camera Kit session to the Surface, you can
// then return it as a LocalStageStream to be published by the Broadcast SDK
val customStream: ImageLocalStageStream = ImageLocalStageStream(surfaceSource)
publishStreams.add(customStream)
```

```
@Override
fun stageStreamsToPublishForParticipant(stage: Stage, participantInfo:
 ParticipantInfo): List<LocalStageStream> = publishStreams
```

背景替換

背景替換是一種攝影機濾鏡，可讓即時串流創作者更改背景。如下圖所示，替換背景包含：

1. 從即時攝影機供稿獲取攝影機影像。
2. 使用 Google ML Kit 將其分割成前景和背景組件。
3. 組合產生的分割遮罩與自訂背景影像。
4. 將其傳遞給自訂影像來源以進行廣播。



Web

本節假設您已熟悉使用 Web 廣播 SDK 發布和訂閱影片。

若要以自訂影像替換即時串流的背景，請使用具有 MediaPipe 影像分割器的自拍分割模型。這是一種機器學習模型，可識別影片影格中的哪些像素位於前景或背景中。然後，您可以使用模型的結果來替換即時串流的背景，方法是將影片供稿中的前景像素複製到代表新背景的自訂影像。

若要整合背景替換與 IVS 即時串流 Web 廣播 SDK，您需要：

1. 安裝 MediaPipe 和 Webpack。(我們的範例使用 Webpack 作為打包工具，但您可以自行選擇任何打包工具。)
2. 建立 `index.html`。
3. 新增媒體元素。
4. 新增指令碼標籤。
5. 建立 `app.js`。
6. 載入自訂背景影像。
7. 建立 `ImageSegmenter` 的執行個體。

8. 將影片供稿轉譯到畫布。
9. 建立背景替換邏輯。
10建立 Webpack 組態檔。
11綁定自己的 JavaScript 檔案。

安裝 MediaPipe 和 Webpack

若要開始，請先安裝 `@mediapipe/tasks-vision` 和 `webpack` npm 套件。以下範例使用 Webpack 作為 JavaScript 打包工具；如果願意，您也可以使用不同的打包工具。

JavaScript

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

請務必更新自己的 `package.json` 將 `webpack` 指定為構建指令碼：

JavaScript

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

建立 index.html

接下來，建立 HTML 樣板並將 Web 廣播 SDK 匯入為指令碼標籤。在下列程式碼中，請務必用您的廣播 SDK 版本取代 `<SDK version>`。

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>
```

新增媒體元素

接下來，在 body 標籤中新增一個影片元素和兩個畫布元素。影片元素會包含即時攝影機供稿，並將用作 MediaPipe 影像分割器的輸入。第一個畫布元素將用於轉譯要廣播的供稿的預覽。第二個畫布元素將用於轉譯要當作背景的自訂影像。由於具有自訂影像的第二個畫布僅用於將像素以編程方式複製到最終畫布的來源，檢視中會隱藏該畫布。

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>
```

新增指令碼標籤

新增指令碼標籤來載入綁定的 JavaScript 檔案，該檔案會包含執行背景替換並將其發布至舞台的程式碼：

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

建立 app.js

接下來建立一個 JavaScript 檔案，獲取在 HTML 頁面中建立的畫布和影片元素的元素物件。匯入 `ImageSegmenter` 和 `FilesetResolver` 模組。`ImageSegmenter` 模組將用於執行分割任務。

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";
```

接下來建立一個名為 `init()` 的函數，從使用者的攝影機擷取 MediaStream，並在每次攝影機影格完成加載時叫用回呼函數。為加入和離開舞台按鈕新增事件接聽程式。

請注意，加入舞台時，我們會傳遞一個名為 `segmentationStream` 的變數。這是從畫布元素擷取的影片串流，其中包含疊加在代表背景的自訂影像上的前景影像。稍後，此自訂串流將用於建立可發布至舞台的 `LocalStageStream` 執行個體。

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();
  });
};
```

載入自訂背景影像

在 init 函數底部新增代碼來呼叫名為 initBackgroundCanvas 的函數，該函數會從本地檔案加載自訂影像並將其轉譯到畫布上。我們將在下一個步驟中定義此函數。將從使用者攝影機擷取的 MediaStream 指派給影片物件。稍後，此影片物件將傳遞給影像分割器。另外，設定一個名為 renderVideoToCanvas 的回呼函數，在影片影格完成加載時叫用。我們將在後續步驟中定義此函數。

JavaScript

```
initBackgroundCanvas();

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

讓我們實現從本地檔案載入影像的 initBackgroundCanvas 函數。此範例使用海灘影像作為自訂背景。包含自訂影像的畫布將被隱藏而不顯示，這是因為您會將其與包含攝影機供稿的畫布元素的前景像素合併。

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);
  };
};
```

建立 ImageSegmenter 的執行個體

接下來建立 ImageSegmenter 的執行個體，該執行個體會分割影像並將結果回傳為遮罩。建立 ImageSegmenter 的執行個體時，您會用到自拍分割模型。

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,
  });
};
```

將影片供稿轉譯到畫布

接下來，建立將影片供稿轉譯到另一個畫布元素的函數。我們需要將影片供稿轉譯到畫布，以便使用 Canvas 2D API 從中提取前景像素。執行此操作時，我們也會將影片影格傳遞給我們的 ImageSegmenter 執行個體，使用 segmentforVideo 方法分割影片影格中的前景和背景。當 segmentforVideo 方法返回時，它會叫用我們的自訂回呼函數 replaceBackground 來執行背景替換。

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);
};
```

建立背景替換邏輯

建立 replaceBackground 函數，將自訂背景影像與攝影機供稿的前景合併以替換背景。該函數會首先從先前建立的兩個畫布元素中，檢索自訂背景影像的基礎像素資料和影片供稿。然後，它反復執行 ImageSegmenter 提供的遮罩，其中指出哪些像素屬於前景。在反復執行遮罩時，它會選擇性地將包含使用者攝影機供稿的像素複製到對應的背景像素資料中。完成後，它會將前景複本上的最終像素資料轉換為背景並繪製到畫布上。

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);
```

```
}
```

作為參考，這裡的完整 app.js 檔案包含了上述所有邏輯：

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();
```

建立一個 Webpack 組態檔

將此組態新增到自己的 Webpack 組態檔來綁定 `app.js`，讓匯入呼叫起作用：

JavaScript

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

綁定自己的 JavaScript 檔案

```
npm run build
```

從包含 `index.html` 的目錄啟動一個簡單的 HTTP 伺服器，然後打開 `localhost:8000` 查看結果：

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

Android

要替換即時串流中的背景，您可以使用 Google ML Kit 的自拍分割 API。自拍分割 API 接受攝影機影像作為輸入，並可傳回遮罩為影像的每個像素提供信賴度分數，指出該像素是在前景中還是背景中。然後，您就能根據信賴度分數從背景影像或前景影像擷取對應的像素顏色。這個過程會持續進行，直到檢查完遮罩中的所有信賴度分數為止。結果會產生一個新的像素顏色陣列，其中包含前景像素與背景影像中像素的組合。

若要整合背景替換與 IVS 即時串流 Android 廣播 SDK，您需要：

1. 安裝 CameraX 程式庫和 Google ML Kit。
2. 初始化樣板變數。
3. 建立自訂影像來源。
4. 管理攝影機影格。
5. 將攝影機影格傳遞給 Google ML Kit。
6. 將攝影機影格前景覆疊到自訂背景上。
7. 將新影像提供給自訂影像來源。

安裝 CameraX 程式庫和 Google ML Kit

要從即時攝影機供稿中提取影像，請使用 Android 的 CameraX 程式庫。要安裝 CameraX 程式庫和 Google ML Kit，請將以下內容新增到模組的 `build.gradle` 檔案中。用最新版本的 CameraX 和 Google ML Kit 程式庫分別替換 `${camerax_version}` 與 `${google_ml_kit_version}`。

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}"
```

匯入下列程式庫：

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
```

初始化樣板變數

初始化 ImageAnalysis 的執行個體和 ExecutorService 的執行個體：

Java

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

在 STREAM_MODE 中初始化一個分割器執行個體：

Java

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

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

建立自訂影像來源

在活動的 onCreate 方法中，建立 DeviceDiscovery 物件的執行個體，並建立一個自訂影像來源。自訂影像來源提供的 Surface 會收到前景疊加在自訂背景影像上的最終影像。然後，您要使用自訂影像來源建立 ImageLocalStageStream 的執行個體。之後，ImageLocalStageStream 的執行個體 (在此範例中名為 filterStream) 就能發布至舞台。如需如何設定舞台的說明，請參閱 IVS Android 廣播 SDK 指南。最後，也要建立一個用於管理攝影機的線程。

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()
```

管理攝影機影格

接下來，建立一個函數來初始化攝影機。此函數使用 CameraX 程式庫從即時攝影機供稿中提取影像。首先，您要建立名為 cameraProviderFuture 的 ProcessCameraProvider 執行個體。該物件表示獲得攝影機提供者的未來結果。然後，您將專案中的影像載入為點陣圖。此範例使用海灘影像作為背景，但您可以使用任何影像。

接著，您將接聽程式新增到 cameraProviderFuture。當攝影機變得可用或在取得攝影機提供者的過程中發生錯誤，此接聽程式機會受到通知。

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))
    }
```

在接聽程式中，建立 ImageAnalysis.Builder 存取即時攝影機供稿中的每個單獨影格。將背壓策略設定為 STRATEGY_KEEP_ONLY_LATEST。這樣可以確保一次僅交付一個攝影機影格進行處理。將每個單獨的攝影機影格轉換為點陣圖，以便您可以提取其像素，並於稍後將其與自訂背景影像合併。

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())
```

將攝影機影格傳遞給 Google ML Kit

接下來，建立 InputImage 並將其傳遞給分割器的執行個體進行處理。可在 ImageAnalysis 執行個體提供的 ImageProxy 中建立 InputImage。只要將 InputImage 提供給分割器，就會回傳一個帶有信賴度分數的遮罩，指示像素處於前景或背景的可能性。這個遮罩還提供寬高屬性，可供您建立一組新的陣列，其中包含先前載入的自訂背景影像的背景像素。

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)
```

將攝影機影格前景覆疊到自訂背景上

有了包含信賴度分數的遮罩、當成點陣圖的攝影機影格以及自訂背景影像中的色彩像素，您就擁有將前景覆疊到自訂背景上所需的一切。接著，就能使用下列參數呼叫 `overlayForeground` 函數：

Java

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

此函數會反复執行遮罩，並檢查信賴度值，從而決定是從背景影像還是攝影機影格取得對應的像素顏色。如果信賴度值表示遮罩中的像素很可能出現在背景中，將從背景影像中獲取相應的像素顏色；否則，將從攝影機影格中獲取相應的像素顏色來建置前景。函數完成對遮罩的反覆處理後，就會使用新的色彩像素陣列建立新的點陣圖並傳回。這個新的點陣圖包含疊加在自訂背景上的前景。

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
        )
    }
```

將新影像提供給自訂影像來源

然後，您可以將新的點陣圖寫入由自訂影像來源提供的 Surface。這會將其廣播到您的舞台。

Java

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

以下是獲取攝影機影格、傳遞給分割器並覆疊在背景上的完整函數：

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 廣播 SDK：行動音訊模式 (即時串流)

音訊品質是任何即時媒體經驗的重要組成部分，而且不存在適用於所有使用案例的通用型音訊組態。為了確保您的使用者在收聽 IVS 即時串流時獲得最佳體驗，我們的行動 SDK 提供了多種預設音訊組態，以及視需要提供的更強大自訂功能。

簡介

IVS 行動廣播 SDK 提供一個 `StageAudioManager` 類別。這個類別被設計成單一接觸點，用於控制兩個平台上的基礎音訊模式。在 Android 上，這可以控制 AudioManager，包括音訊模式、音訊來源、內容類型、使用情況和通訊裝置。在 iOS 上，它可控制應用程式 AVAudioSession，以及是否啟用 voiceProcessing。

重要事項：當 IVS 即時廣播 SDK 啟用時，請勿與 `AVAudioSession` 或 `AudioManager` 直接互動。因為這可能會導致音訊遺失，或是從錯誤的裝置錄製、播放音訊。

在建立第一個 `DeviceDiscovery` 或 `Stage` 物件之前，必須先設定 `StageAudioManager` 類別。

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
```

如果在初始化 DeviceDiscovery 或 Stage 執行個體之前，未在 StageAudioManager 上作任何設定，則會自動套用 VideoChat 預設值。

音訊模式預設值

即時廣播 SDK 提供三組預設值，每組都是針對常用案例量身打造，如下所述。每組預設值都涵蓋五個關鍵類別，好將各組預設值區分開來。

視訊聊天

這是預設值，專為本機裝置與其他參與者進行即時對話設計。

類別	Android	iOS
回音消除	已啟用	已啟用
音量鍵	通話音量	通話音量
麥克風選擇	受作業系統限制。USB 麥克風可能無法使用。	受作業系統限制。USB 和藍牙麥克風可能無法使用。

類別	Android	iOS
		同時處理輸入和輸出的藍牙耳機應能正常工作，例如 AirPods。
音訊輸出	任何輸出裝置都應能正常工作。	受作業系統限制。有線耳機可能無法使用。
音訊品質	中/低。聽起來像是在講電話，而非播放媒體。	中/低。聽起來像是在講電話，而非播放媒體。

僅限訂閱

此預設值是為您訂閱其他發布參與者的計畫而設計，並非用於發布自己。它專注於音訊品質且支持所有可用的輸出裝置。

類別	Android	iOS
回音消除	已停用	已停用
音量鍵	媒體音量	媒體音量
麥克風選擇	不適用。此預設值不是為發布而設計。	不適用。此預設值不是為發布而設計。
音訊輸出	任何輸出裝置都應能正常工作。	任何輸出裝置都應能正常工作。
音訊品質	高。任何媒體類型都應該能清晰地播放，包括音樂。	高。任何媒體類型都應該能清晰地播放，包括音樂。

Studio

此預設值是為了高品質的訂閱而設計，同時也保持了發布能力。它需要錄製和播放硬體才能提供回音消除功能。這裡的一個使用案例就是使用 USB 麥克風和有線耳機。SDK 將保持最高品質的音訊，同時依靠這些裝置的物理分離防止回音。

類別	Android	iOS
回音消除	已停用	已停用
音量鍵	大多數情況下的媒體音量。連接藍牙麥克風時的通話音量。	媒體音量
麥克風選擇	任何麥克風都應該可用。	任何麥克風都應該可用。
音訊輸出	任何輸出裝置都應能正常工作。	任何輸出裝置都應能正常工作。
音訊品質	高。雙方應該都能發送音樂並在另一側清晰地聽到。 連接藍牙耳機後，音訊品質可能會因為啟用了藍牙 SCO 模式而下降。	高。雙方應該都能發送音樂並在另一側清晰地聽到。 連接藍牙耳機後，根據耳機的不同，音訊品質可能會因為啟用了藍牙 SCO 模式而下降。

進階使用案例

除了預設值之外，iOS 和 Android 即時串流廣播 SDK 都允許設定基礎平台音訊模式：

- 在 Android 上，設定 AudioSource、Usage 和 ContentType。
- 在 iOS 上，使用 AVAudioSession.Category、AVAudioSession.CategoryOptions、AVAudioSession.Mode，以及在發布時切換是否啟用 voice processing 的功能。

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
```

在 Android 系統上以藍牙發布

在滿足下列條件時，SDK 會自動回復為 Android 上的 `VIDEO_CHAT` 預設值：

- 指派的組態不會使用 `VOICE_COMMUNICATION` 使用率值。
- 藍牙麥克風已連接至裝置。
- 本機參與者正在發布至階段。

這是 Android 作業系統關於如何使用藍牙耳機錄製音訊的限制。

與其他 SDK 整合

由於 iOS 和 Android 的每個應用程式均僅支持一種主動音訊模式，因此如果您的應用程式使用多種需要控制音訊模式的 SDK，則通常會遇到衝突。當您遇到這些衝突時，有一些常見的解決策略可嘗試，詳情如下所述。

對齊音訊模式值

使用 IVS SDK 的進階音訊組態選項或其他 SDK 功能，讓兩個 SDK 的基礎值對齊。

Agora

iOS

在 iOS 上，告訴 Agora SDK 保持 `AVAudioSession` 主動，將阻止它被 IVS 即時串流廣播 SDK 使用時遭到停用。

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

Android

避免在 `RtcEngine` 上呼叫 `setEnableSpeakerphone`，並於使用 IVS 即時串流廣播 SDK 發布時呼叫 `enableLocalAudio(false)`。當 IVS SDK 沒有在發布時，您可以再次呼叫 `enableLocalAudio(true)`。

搭配 IVS 即時串流使用 Amazon EventBridge

您可以使用 Amazon EventBridge 監控您的 Amazon Interactive Video Service (IVS) 串流。

Amazon IVS 將有關串流狀態的變更事件傳送至 Amazon EventBridge。交付的所有事件都是有效的。無論如何會竭盡全力傳送事件，這表示並不能保證：

- 事件被交付 – 可能會發生指定的事件 (例如，發布的參與者)，但 Amazon IVS 可能不會將相應的事件傳送至 EventBridge。Amazon IVS 會在放棄前嘗試用數小時交付事件。
- 交付的事件將在指定的時間範圍內送達：您可能會收到幾個小時前的事件。
- 按照順序交付的事件：事件可能會失序，特別是如果在短時間內傳送它們。例如，您可能會在發布參與者之前看到未發布的參與者。

雖然事件很少會遺失、遲到或失序，但如果您撰寫依賴於通知事件順序或存在的業務關鍵程式，則應該處理這些可能性。

您可以針對以下任何事件建立 EventBridge 規則。

事件類型	事件	傳送時機...
IVS 合成狀態變更	目的地失敗	輸出至目的地嘗試失敗。例如，廣播至頻道失敗，因為沒有串流金鑰或正在進行其他廣播。
IVS 合成狀態變更	目的地啟動	輸出至目的地成功啟動。
IVS 合成狀態變更	目的地結束	輸出至目的地完成。
IVS 合成狀態變更	目的地重新連線	輸出至目的地中斷且正在嘗試重新連線。
IVS 合成狀態變更	工作階段啟動	已建立合成工作階段。當合成程序管道成功初始化時，會觸發此事件。此時，合成管道已成功訂閱階段，正在接收媒體並且能夠合成視訊。
IVS 合成狀態變更	工作階段結束	合成工作階段已完成。

事件類型	事件	傳送時機...
IVS 合成狀態變更	工作階段失敗	合成管道無法初始化，因為階段資源無法使用或出現任何其他內部錯誤。
IVS 階段更新	參與者已發布	參與者開始發布至階段。
IVS 階段更新	參與者未發布	參與者已停止發布至階段。

為 Amazon IVS 建立 Amazon EventBridge 規則

您可以建立針對 Amazon IVS 發出的事件而觸發的規則。遵循 Amazon EventBridge 使用者指南中的在 Amazon EventBridge 中建立規則的步驟。選取服務時，請選擇 Interactive Video Service (IVS)。

範例：合成狀態變更

目的地失敗：輸出至目的地嘗試失敗時，即會傳送此事件。例如，廣播至頻道失敗，因為沒有串流金鑰或正在進行其他廣播。

```
{
   "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"
   }
}
```

目的地啟動：輸出至目的地成功啟動時，即會傳送此事件。

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

目的地結束：輸出至目的地完成時，即會傳送此事件。

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

目的地重新連線：輸出至目的地中斷且正在嘗試重新連線時，即會傳送此事件。

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

工作階段啟動：合成工作階段建立時，即會傳送此事件。當合成程序管道成功初始化時，會觸發此事件。此時，合成管道已成功訂閱階段，正在接收媒體並且能夠合成視訊。

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

工作階段結束：合成工作階段完成並刪除所有資源時，即會傳送此事件。

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

工作階段失敗：階段資源無法使用、階段中沒有參與者或任何其他內部錯誤導致合成管道初始化失敗時，會傳送此事件。

```
{
   "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"
   }
}
```

範例：階段更新

階段更新事件包括事件名稱 (將事件分類) 和有關該事件的中繼資料。中繼資料包括觸發事件的參與者 ID、關聯的階段和工作階段 ID 以及使用者 ID。

參與者已發布：當參與者開始發布至階段時，會傳送此事件。

```
{
```

```
    "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"
    }
}
```

參與者未發布：當參與者已停止發布至階段時，會傳送此事件。

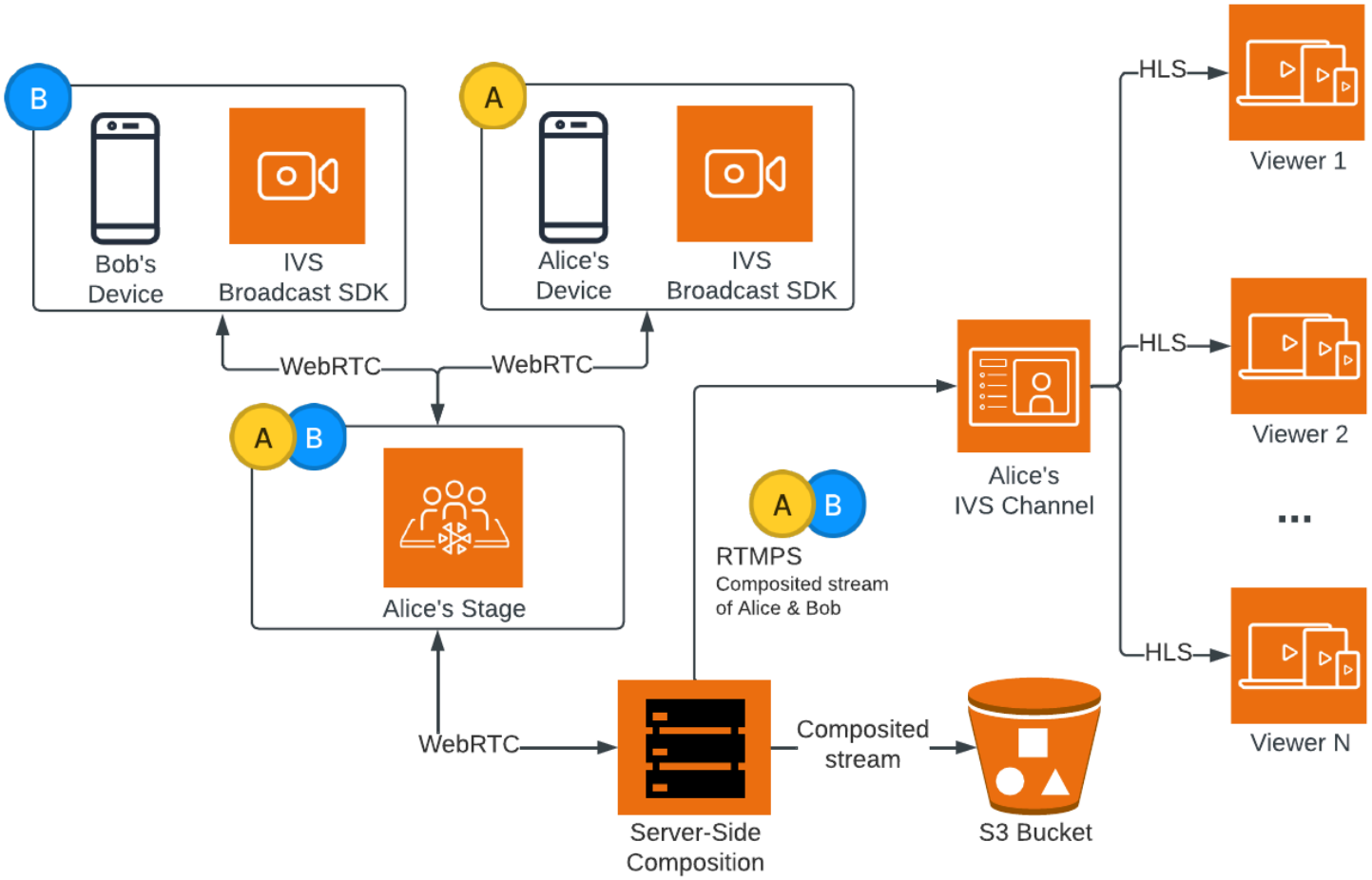
```
{
    "version": "0",
    "id": "12345678-1a23-4567-a1bc-1a2b34567890",
    "detail-type": "IVS Stage Update",
    "source": "aws.ivs",
    "account": "123456789012",
    "time": "2020-06-23T20:12:36Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij"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

伺服器端合成 (即時串流)

伺服器端合成使用 IVS 伺服器來混合所有舞台參與者的音訊和視訊，然後將此混合視訊傳送至 IVS 頻道 (例如觸及更多觀眾) 或 S3 儲存貯體。伺服器端合成會透過舞台主區域中的 IVS 控制平面端點叫用。

使用伺服器端合成廣播或錄製舞台會提供許多好處，可以讓使用者獲得高效可靠的雲端型影片工作流程，使其成為一個吸引人的選擇。

下圖說明伺服器端合成的運作方式：



優勢

與用戶端合成相比，伺服器端合成具有以下優點：

- 減少用戶端負載：透過伺服器端合成，處理和合併音訊與視訊來源的負擔會從個別用戶端裝置轉移到伺服器本身。伺服器端合成可消除用戶端裝置使用其 CPU 和網路資源來合成檢視並將其傳輸到 IVS

的需求。這表示觀眾可以觀看廣播，而其裝置無需處理資源密集型任務，這可以改善電池壽命和更流暢的觀看體驗。

- 一致的品質：伺服器端合成可讓您精確控制最終串流的品質、解析度和位元速率。這可確保所有觀眾獲得一致的觀看體驗，無論其個別裝置的功能為何。
- 彈性：透過將合成程序集中在伺服器上，廣播會變得更加穩定。即使發布者裝置遇到技術限制或波動，伺服器也可以適應並為所有觀眾提供更流暢的串流。
- 頻寬效率：由於伺服器會處理合成，因此舞台發布者不必花費額外頻寬將視訊廣播到 IVS。

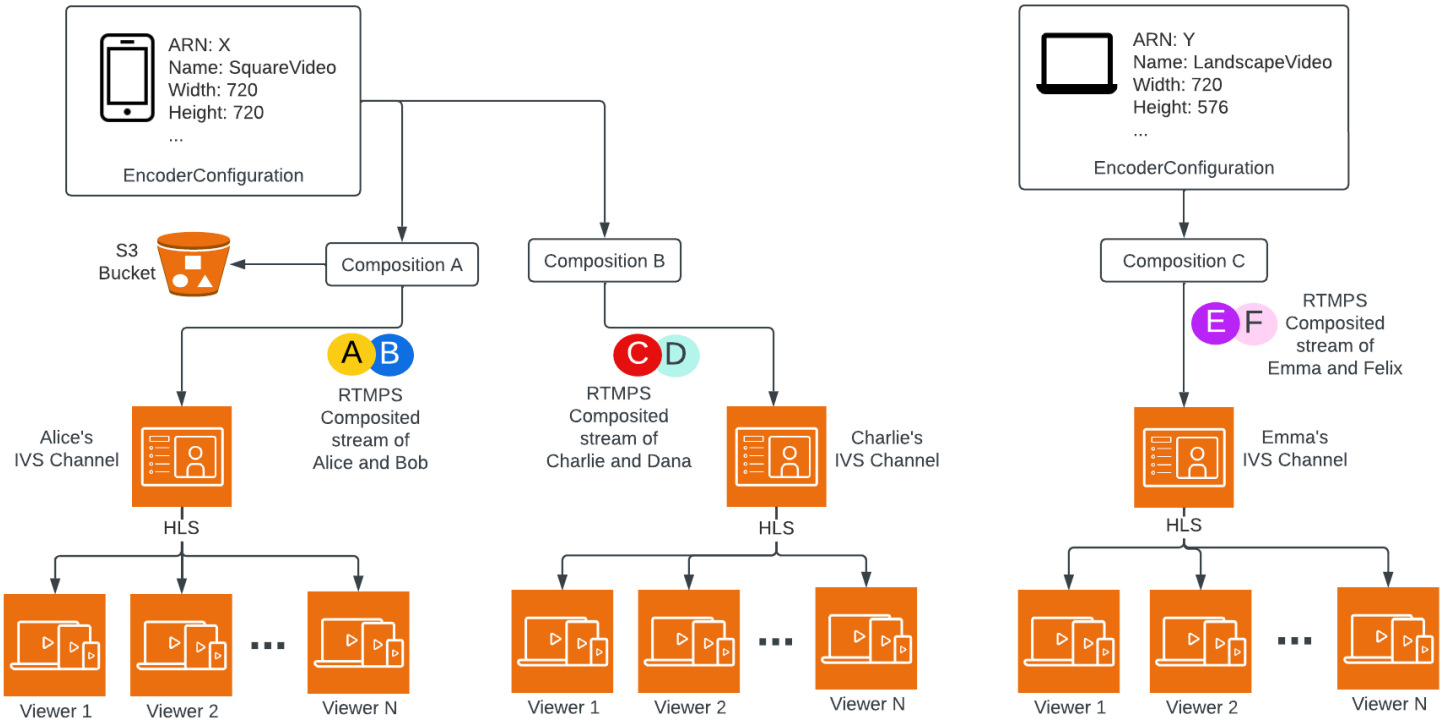
若要將舞台廣播到 IVS 頻道，您可以執行用戶端合成；請參閱《IVS 低延遲串流使用者指南》中的在 IVS 串流上啟用多位主持人。

IVS API

伺服器端合成使用下列關鍵 API 元素：

- EncoderConfiguration 物件允許您自訂要生成的影片格式 (高度、寬度、位元速率和其他串流參數)。您可以在每次呼叫 StartComposition 端點時重複使用 EncoderConfiguration。
- Composition 端點會追蹤視訊合成，並輸出至 IVS 頻道。
- StorageConfiguration 會追蹤記錄合成的 S3 儲存貯體。

若要使用伺服器端合成，您需要建立一個 EncoderConfiguration 並在呼叫 StartComposition 端點時連接。在此範例中，SquareVideo EncoderConfiguration 用於兩種合成：

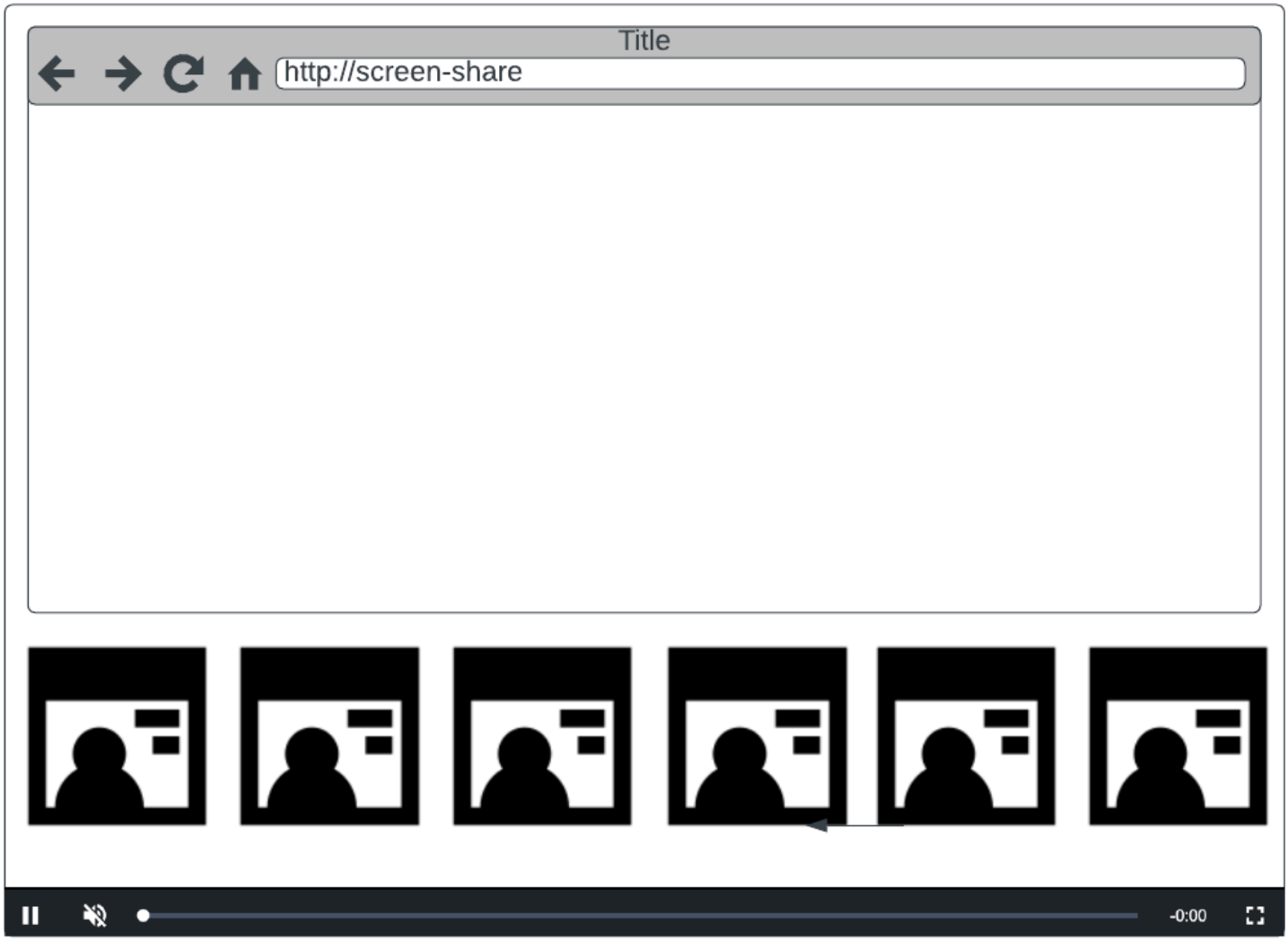


如需完整資訊，請參閱 IVS Real-Time Streaming API Reference。

版面配置

根據預設，伺服器端合成功能會使用格線版面配置，將舞台參與者排列在同等大小的插槽中：

此版面配置提供客戶設定及叫用精選插槽的選項。精選插槽位於主要畫面上，其他參與者會在其下方同樣大小的插槽中顯示：



注意：舞台發布者在伺服器端合成上所支援的最大解析度為 1080p。如果發布者傳送的視訊高於 1080p，則發布者會轉譯為純音訊參與者。

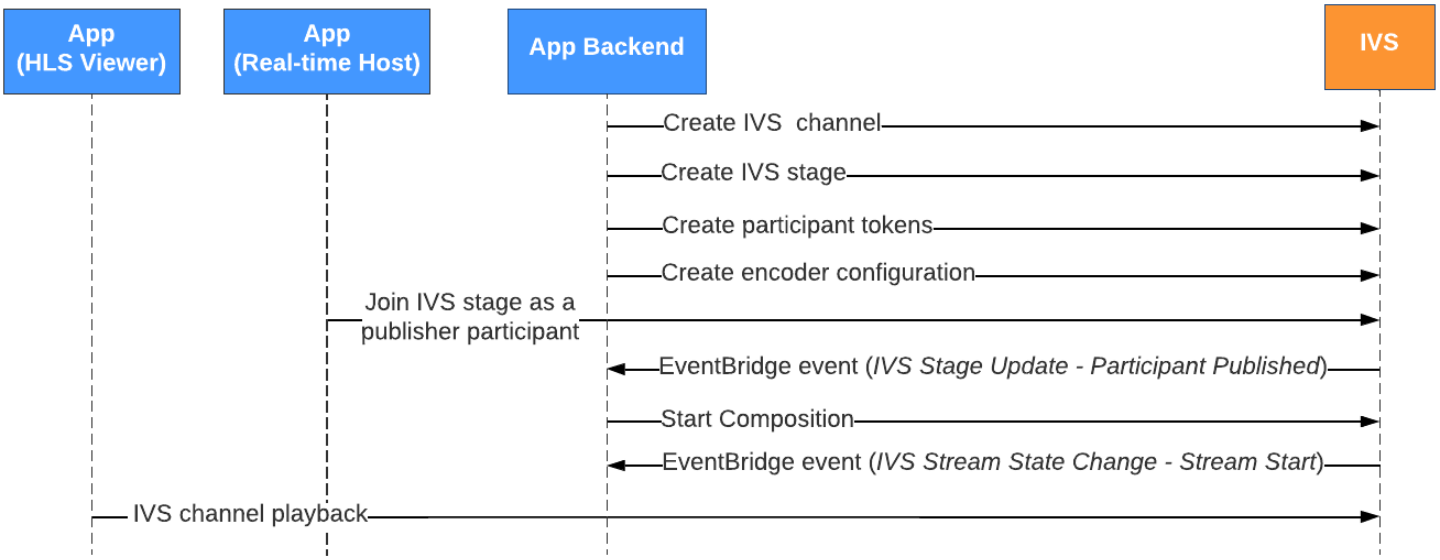
入門

先決條件

若要使用伺服器端合成，您必須具有包含作用中發布者的舞台，並使用 IVS 頻道和/或 S3 儲存貯體作為合成目的地。下面，我們描述了一種可能的工作流程，該工作流程使用 EventBridge 事件啟動合成，該合成會在參與者發布時將舞台廣播到 IVS 頻道。您也可以根據自己的應用程式邏輯啟動和停止合成。請參閱複合錄製中的另一個範例，其中展示了如何使用伺服器端合成將舞台直接錄製到 S3 儲存貯體。

1. 建立一個 IVS 頻道。請參閱開始使用 Amazon IVS 低延遲串流功能。

2. 為每個發布者建立 IVS 舞台和參與者權杖。
3. 建立一個 EncoderConfiguration。
4. 加入並發布到舞台。(請參閱 Web 版、Android 版和 iOS 版即時串流廣播 SDK 指南中的「發布與訂閱」章節。)
5. 當您收到參與者已發布的 EventBridge 事件時，請呼叫 StartComposition。
6. 等待幾秒鐘，然後在頻道播放中查看複合檢視。



注意：在舞台上的發布者參與者閒置 60 秒後，合成會執行自動關閉。此時，合成會終止並轉換為 STOPPED 狀態。合成會在保持 STOPPED 狀態幾分鐘後自動刪除。

CLI 說明

使用 AWS CLI 是進階選項，需要您先在機器上下載並設定 CLI。如需詳細資訊，請參閱《AWS 命令列界面使用者指南》。

您目前可以使用 CLI 來建立和管理資源。合成端點位於 `ivs-realtime` 命名空間下。

建立 EncoderConfiguration 資源

EncoderConfiguration 是一個物件，允許您自訂生成影片的格式 (高度、寬度、位元速率和其他串流參數)。您可以在每次呼叫合成端點時重複使用 EncoderConfiguration，如下一個步驟所述。

下列命令會建立 EncoderConfiguration 資源，可設定伺服器端視訊合成參數，例如視訊位元速率、影格率和解析度：

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

回應為：

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

啟動合成

使用上述回應中提供的 EncoderConfiguration ARN 建立您的合成資源：

```
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"}}]'
```

回應將顯示合成建立時即處於 STARTING 狀態。只要合成開始發布合成，狀態就會轉換為 ACTIVE。(您可以透過呼叫 ListCompositions 或 GetComposition 端點來查看狀態。)

只要合成處於 ACTIVE 狀態，IVS 舞台的複合檢視就可以在 IVS 頻道上出現，使用 ListCompositions：

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

回應為：

```
{
   "compositions": [
```

```
      {
         "arn": "arn:aws:ivs:us-east-1:927810967299:composition/YVoaXkKdEdRP",
         "destinations": [
            {
               "id": "bD9rRoN91fHU",
               "startTime": "2023-09-21T15:38:39+00:00",
               "state": "ACTIVE"
            }
         ],
         "stageArn": "arn:aws:ivs:us-east-1:927810967299:stage/8faHz1SQp0ik",
         "startTime": "2023-09-21T15:38:37+00:00",
         "state": "ACTIVE",
         "tags": {}
      }
   ]
}
```

注意：您必須讓發布者參與者主動發布至舞台，才能讓合成保持作用中狀態。如需詳細資訊，請參閱 Web 版、Android 版和 iOS 版即時串流廣播 SDK 指南中的「發布與訂閱」章節。您必須為每個參與者建立不同的舞台權杖。

啟用螢幕共用

若要使用固定螢幕共用版面配置，請按照以下步驟操作。

建立 EncoderConfiguration 資源

下列命令會建立 EncoderConfiguration 資源，可設定伺服器端合成參數 (視訊位元速率、影格率和解析度)。

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

建立具有 screen-share 屬性的舞台參與者權杖。由於我們將 screen-share 指定為 featured 插槽的名稱，因此我們需要建立一個舞台權狀，並將 screen-share 屬性設定為 true：

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

回應為：

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

啟動合成

若要使用螢幕共用功能啟動合成，我們使用以下命令：

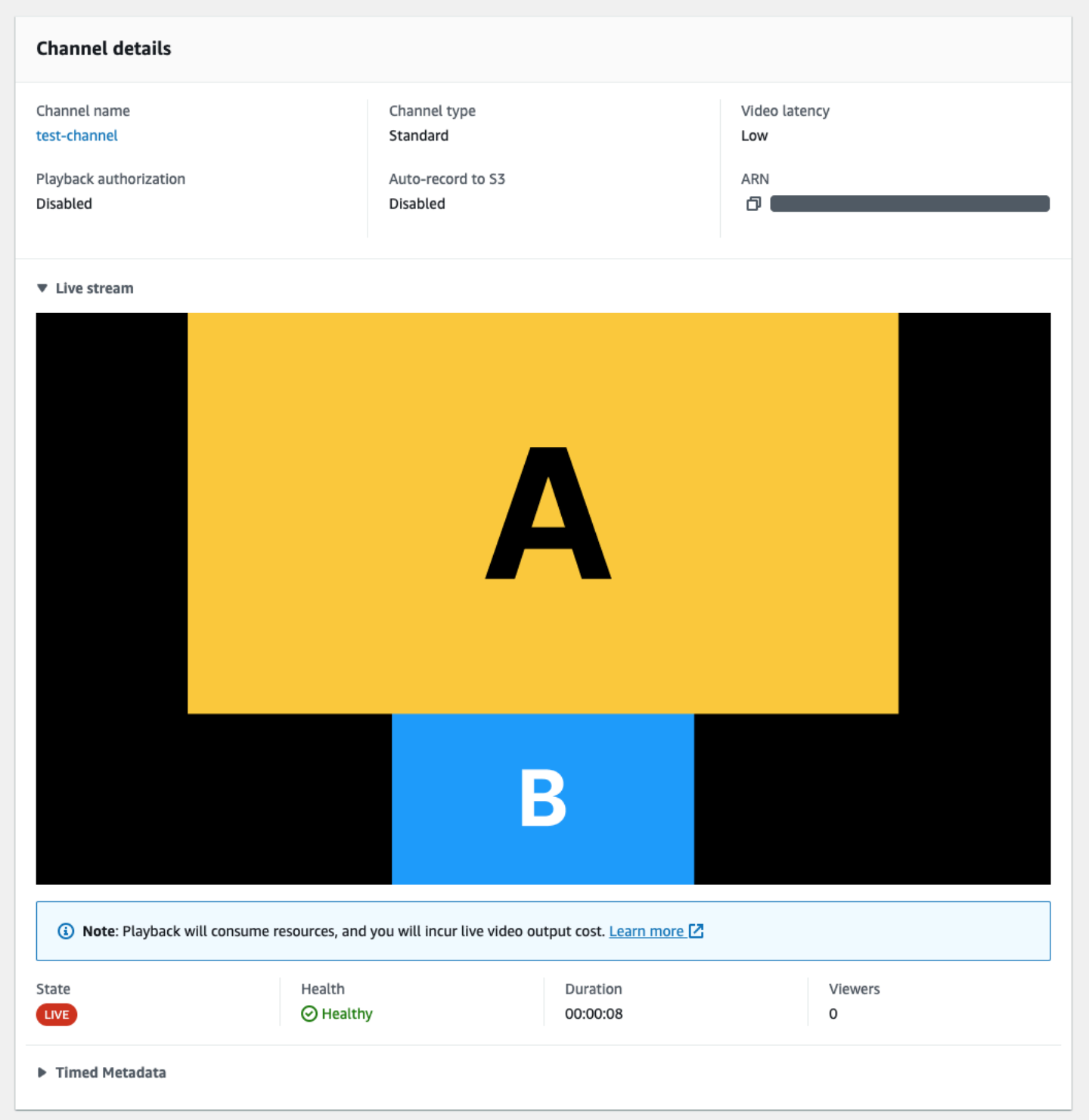
```
aws ivs-realtime start-composition --stage-arn "arn:aws:ivs:us-
east-1:927810967299:stage/8faHz1SQp0ik" --destinations  '[{"channel": {"channelArn":
 "arn:aws:ivs:us-east-1:927810967299:channel/DOlMW4dfMR8r", "encoderConfigurationArn":
 "arn:aws:ivs:us-east-1:927810967299:encoder-configuration/DEkQHWPVaOwO"}}]' --layout
 "grid={featuredParticipantAttribute=screen-share}"
```

回應為：

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

當舞台參與者 E813MFklPWLF 加入舞台時，該參與者的視訊將顯示在精選插槽中，其餘舞台發布者將會在插槽下方轉譯：



停止合成

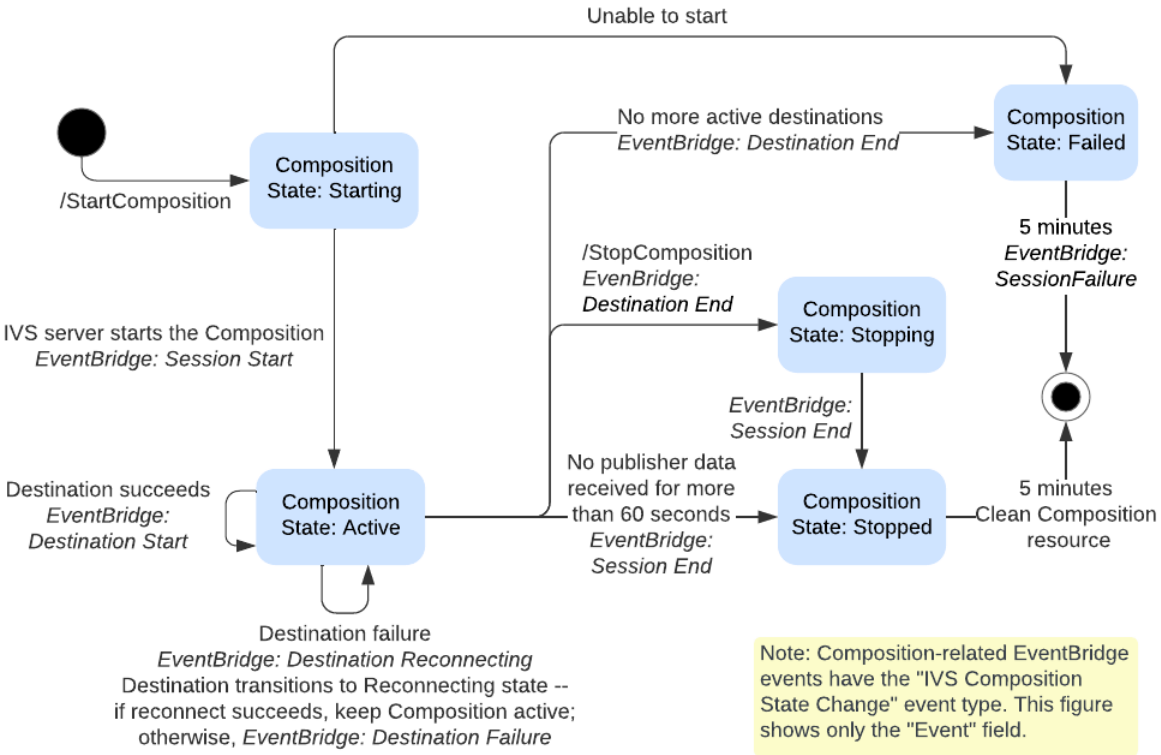
若要在任何時間點停止合成，請呼叫 StopComposition 端點：

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

合成生命週期

參閱下方圖表，了解合成的狀態轉換。合成的生命週期大致如下：

1. 當使用者呼叫 StartComposition 端點時，會建立合成資源
2. 一旦 IVS 成功啟動合成，就會傳送「IVS 合成狀態變更 (工作階段啟動)」EventBridge 事件。如需有關事件的詳細資訊，請參閱搭配 IVS 即時串流使用 EventBridge。
3. 只要合成處於作用中狀態，可能會發生以下情況：
   - 使用者停止合成：如果呼叫 StopComposition 端點，IVS 會起始正常關閉合成，並傳送「目的地結束」事件，然後傳送「工作階段結束」事件。
   - 合成執行自動關閉：如果沒有參與者主動發布到 IVS 舞台，則合成會在 60 秒後自動完成，並傳送 EventBridge 事件。
   - 目的地失敗：如果目的地意外失敗 (例如，IVS 頻道遭到刪除)，目的地會轉換為 RECONNECTING 狀態，並傳送「目的地重新連線」事件。如果無法復原，IVS 會將目的地轉換為 FAILED 狀態，並傳送「目的地失敗」事件。如果合成中至少有一個目的地處於作用中狀態，IVS 會保持作用中狀態。
4. 只要合成處於 STOPPED 或 FAILED 狀態，會在五分鐘後自動清理。(之後無法再被 ListCompositions 或 GetComposition 擷取。)


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.

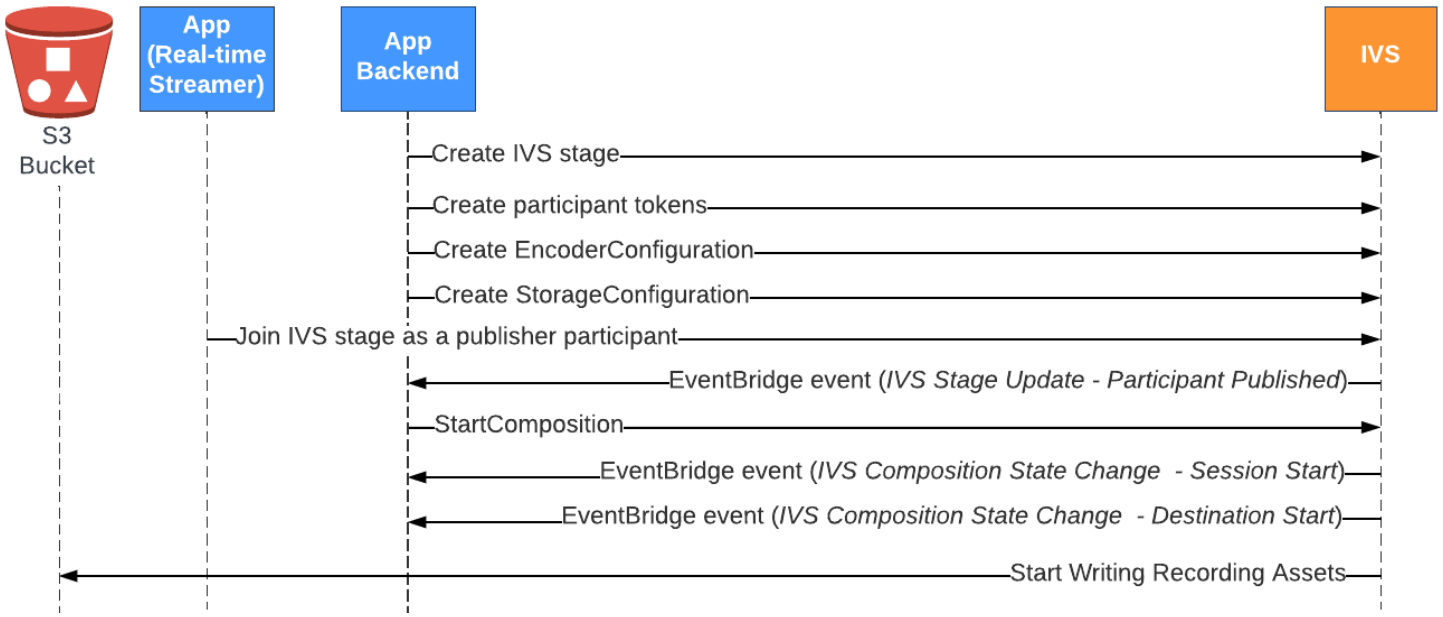
複合錄製 (即時串流)

本文件說明如何在伺服器端合成中使用複合錄製功能。複合錄製可讓您產生 IVS 階段的 HLS 錄製檔，方法是使用 IVS 伺服器將所有階段發布者有效地合併至一個檢視，然後將產生的影片儲存到 S3 儲存貯體。

必要條件

若要使用複合錄製，您必須具有作用中發行者的階段，以及 S3 儲存貯體作為錄製目的地。下面，我們描述了一種使用 EventBridge 事件將組成記錄到 S3 存儲桶的可能工作流程。您也可以根據自己的應用程式邏輯啟動和停止合成。

1. 為每個發布者建立 IVS 階段和參與者權杖。
2. 創建一個 EncoderConfiguration（表示如何呈現錄製的視頻的對象）。
3. 創建一個 S3 存儲桶和一個 StorageConfiguration（將存儲記錄內容的位置）。
4. 加入並發布到階段。
5. 當您收到「參與者已發佈」EventBridge 事件時，請StartComposition以 S3 DestinationConfiguration 物件作為目的地呼叫
6. 幾秒鐘後，您應該能看到 HLS 片段已被保留在 S3 儲存貯體中。



注意：在階段上的發布者參與者閒置 60 秒後，合成會執行自動關閉。此時，合成會終止並轉換為 STOPPED 狀態。合成會在保持 STOPPED 狀態幾分鐘後自動刪除。如需詳細資訊，請參閱「伺服器端合成」中的合成生命週期。

複合錄製範例： StartComposition 使用 S3 儲存貯體目的地

以下範例顯示StartComposition端點的典型呼叫，指定 S3 作為構成的唯一目的地。只要合成變更為 ACTIVE 狀態，影片片段和中繼資料就會開始寫入 storageConfiguration 物件指定的 S3 儲存貯體。若要建立具有不同配置的合成，請參閱伺服器端合成中的「版面配置」和 IVS Real-Time Streaming API Reference。

請求

```
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"
}
```

回應

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

存在於 StartComposition 響應中的`recordingPrefix`字段可用於確定記錄內容的存儲位置。

錄製內容

當構成轉換到某個`ACTIVE`狀態時，您將開始看到 HLS 視訊片段和中繼資料檔案寫入呼叫 StartComposition時提供的 S3 儲存貯體。這些內容可用於後續處理或作為隨需影片播放。

請注意，合成上線之後，即會發出「IVS 合成狀態變更」事件，並且可能需要一點時間才能寫入清單檔案和影片片段。建議您只在收到「IVS 合成狀態變更 (工作階段結束)」事件之後，才播放或處理錄製的串流。如需詳細資訊，請參閱搭 EventBridge 配 IVS 即時串流使用。

以下是即時 IVS 工作階段錄製的範例目錄結構和內容：

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

events 資料夾包含對應於錄製事件的中繼資料檔案。JSON 中繼資料檔案會在錄製開始、成功結束或以失敗結束時產生：

- events/recording-started.json
- events/recording-ended.json
- events/recording-failed.json

指定的 events 資料夾將包含 recording-started.json 以及 recording-ended.json 或 recording-failed.json。

它們包含與錄製的工作階段及其輸出格式相關的中繼資料。JSON 的詳細資訊如下所示。

media 資料夾包含支援的媒體內容。hls 子資料夾包含合成工作階段期間產生的所有媒體和清單檔案，並且可以使用 IVS 播放器播放。HLS 清單檔案位於 multivariant.m3u8 資料夾中。

儲存貯體政策 StorageConfiguration

建立 StorageConfiguration 物件時，IVS 將可以存取將內容寫入指定 S3 儲存貯體。該存取權透過修改 S3 儲存貯體的政策來授予。如果以移除 IVS 存取權的方式變更儲存貯體的政策，則進行中和新的錄製將會失敗。

以下範例顯示允許 IVS 寫入 S3 儲存貯體的 S3 儲存貯體政策：

```
{
    "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 中繼資料檔案

此中繼資料為 JSON 格式。其中包含下列資訊：

欄位	類型	必要	描述
`stage_arn`	string	是	作為合成來源而使用的階段 ARN。
`media`	object	是	包含此錄製可用媒體內容之列舉物件的物件。有效值：`"hls"`。
`hls`	object	是	說明 Apple HLS 格式輸出的列舉欄位。
`duration_ms`	integer	有條件	錄製的 HLS 內容的持續時間，以毫秒為單位。當 `recording_status` 為 `"RECORDING_ENDED"` 或 `"RECORDING_ENDED_WITH_FAILURE"` 時才可用。如果在任何錄製完成之前發生失敗，它為 0。
`path`	string	是	儲存 HLS 內容之 S3 字首的相對路徑。

欄位	類型	必要	描述
playlist	string	是	HLS 主播放清單檔案的名稱。
renditions	object	是	中繼資料物件的轉譯 (HLS 變體) 陣列。總是至少有一個轉譯。
path	string	是	為此轉譯儲存 HLS 內容之 S3 字首的相對路徑。
playlist	string	是	此轉譯的媒體播放清單檔案名稱。
resolution_height	int	有條件	編碼影片的像素解析度高度。轉譯包含影片軌道時才可用。
resolution_width	int	有條件	編碼影片的像素解析度寬度。轉譯包含影片軌道時才可用。
recording_ended_at	string	有條件	錄製結束時，RFC 3339 UTC 時間戳記。當 recording_status 為 "RECORDING_ENDED" 或 "RECORDING_ENDED_WITH_FAILURE" 時才可用。 recording_started_at 和 recording_ended_at 是這些事件生成時的時間戳記，未必與 HLS 影片區段時間戳記完全相符。若要準確判斷錄製的持續時間，請使用 duration_ms 欄位。
recording_started_at	string	有條件	錄製開始時，RFC 3339 UTC 時間戳記。如果 recording_status 為 RECORDING_START_FAILED ，則無法使用此選項。 請參閱上面的 recording_ended_at 備註。

欄位	類型	必要	描述
recording_status	string	是	錄製的狀態。有效值："RECORDING_STARTED" 、"RECORDING_ENDED" 、"RECORDING_START_FAILED" 、"RECORDING_ENDED_WITH_FAILURE" 。
recording_status_message	string	有條件	狀態的描述性資訊。當 recording_status 為 "RECORDING_ENDED" 或 "RECORDING_ENDED_WITH_FAILURE" 時才可用。
version	string	是	中繼資料結構描述的版本。

範例：recording-started.json

```
{
  "version": "v1",
  "stage_arn": "arn:aws:ivs:ap-northeast-1:123456789012:stage/aAbBcCdDeE12",
  "recording_started_at": "2023-11-01T06:01:36Z",
  "recording_status": "RECORDING_STARTED",
  "media": {
    "hls": {
      "path": "media/hls",
      "playlist": "multivariant.m3u8",
      "renditions": [
        {
          "path": "720p30-abcdeABCDE12",
          "playlist": "playlist.m3u8",
          "resolution_width": 1280,
          "resolution_height": 720
        }
      ]
    }
  }
}
```

範例：recording-ended.json

```
{
  "version": "v1",
  "stage_arn": "arn:aws:ivs:ap-northeast-1:123456789012:stage/aAbBcCdDeE12",
  "recording_started_at": "2023-10-27T17:00:44Z",
  "recording_ended_at": "2023-10-27T17:08:24Z",
  "recording_status": "RECORDING_ENDED",
  "media": {
    "hls": {
      "duration_ms": 460315,
      "path": "media/hls",
      "playlist": "multivariant.m3u8",
      "renditions": [
        {
          "path": "720p30-abcdeABCDE12",
          "playlist": "playlist.m3u8",
          "resolution_width": 1280,
          "resolution_height": 720
        }
      ]
    }
  }
}
```

範例：recording-failed.json

```
{
  "version": "v1",
  "stage_arn": "arn:aws:ivs:ap-northeast-1:123456789012:stage/aAbBcCdDeE12",
  "recording_started_at": "2023-10-27T17:00:44Z",
  "recording_ended_at": "2023-10-27T17:08:24Z",
  "recording_status": "RECORDING_ENDED_WITH_FAILURE",
  "media": {
    "hls": {
      "duration_ms": 460315,
      "path": "media/hls",
      "playlist": "multivariant.m3u8",
      "renditions": [
        {
          "path": "720p30-abcdeABCDE12",
          "playlist": "playlist.m3u8",
```

```
          "resolution_width": 1280,
          "resolution_height": 720
        }
      ]
    }
  }
}
```

從私人儲存貯體播放錄製的內容

預設情況下，錄製內容為私有；因此，直接使用 S3 URL 無法存取這些物件。如果您嘗試使用 IVS 播放器或其他播放器開啟 HLS 多重變數播放清單 (m3u8 檔案) 進行播放，就會收到錯誤訊息 (例如「您沒有存取所請求資源的許可」)。相反地，您可以使用 Amazon CloudFront CDN (內容交付網路) 播放這些檔案。

CloudFront 可以配置發行版來提供私有存儲桶中的內容。通常，這比具有公開可訪問的存儲桶更好，其中讀取繞過提供的控件。 CloudFront您可以透過建立原始存取控制 (OAC) 來設定要從私有儲存貯體提供服務的發行版，這是在私有原始值區上具有讀取權限的特殊 CloudFront 使用者。您可以在建立散發之後，透過 CloudFront 主控台或 API 建立 OAC。請參閱 Amazon CloudFront 開發人員指南中的建立新的來源存取控制。

在啟用 CORS CloudFront 的情況下使用設定播放

此範例涵蓋開發人員如何在啟用 CORS 的情況下設定發行 CloudFront 版，以便從任何網域播放其錄製檔。這在開發階段特別有用，但您可以修改以下範例以符合生產需求。

步驟 1：建立 S3 儲存貯體

建立將用來存放錄製檔的 S3 儲存貯體。請注意，儲存貯體必須位在您用於 IVS 工作流程的相同區域中。

將 CORS 許可政策新增至儲存貯體：

1. 在 AWS Console 中，前往 S3 儲存貯體許可索引標籤。
2. 複製下面的 CORS 政策，並將其粘貼到跨來源資源共用 (CORS)下。這將啟用 S3 儲存貯體上的 CORS 存取。

```
[
    {
```

```
        "AllowedHeaders": [
            "*"
        ],
        "AllowedMethods": [
            "PUT",
            "POST",
            "DELETE",
            "GET"
        ],
        "AllowedOrigins": [
            "*"
        ],
        "ExposeHeaders": [
            "x-amz-server-side-encryption",
            "x-amz-request-id",
            "x-amz-id-2"
        ]
    }
]
```

步驟 2：建立 CloudFront 分發

請參閱CloudFront 開發人員指南中的建立發 CloudFront 行版。

使用 AWS Console 輸入以下資訊：

對於此欄位	選擇此項目
原始網域	在上一步建立的 S3 儲存貯體
原始存取	原始存取控制設定 (建議使用)，使用預設參數
預設的快取行為：檢視器通訊協定	重新引導 HTTP 到 HTTPS
預設快取行為：允許的 HTTP 方法	GET、HEAD 和 OPTIONS
預設快取行為：快取索引鍵和原始請求	CachingDisabled 政策
預設快取行為：原始要求政策	CORS-S3Origin
預設快取行為：回應標頭政策	SimpleCORS

對於此欄位	選擇此項目
Web 應用程式防火牆	啟用安全保護

然後保存 CloudFront 分發。

步驟 3：設定 S3 儲存貯體政策

1. 刪除您 StorageConfiguration 為 S3 儲存貯體設定的任何項目。這將移除為該儲存貯體建立政策時自動新增的任何儲存貯體政策。
2. 轉到您的 CloudFront 分發，確保所有分發字段都處於上一步中定義的狀態，並複製存儲桶策略（使用複製策略按鈕）。
3. 前往您的 S3 儲存貯體。在許可索引標籤上選取編輯儲存貯體政策，然後貼上您在上一步複製的儲存貯體政策。完成此步驟之後，值區政策應該僅具有該 CloudFront 策略。
4. 創建一個 StorageConfiguration，指定 S3 存儲桶。

建立之 StorageConfiguration 後，您會在 S3 儲存貯體政策中看到兩個項目，一個允許讀CloudFront 取內容，另一個項目允許 IVS 寫入內容。具有 CloudFront 和 IVS 存取權的最終儲存貯體政策範例顯示於範例：具有 CloudFront 和 IVS 存取的 S3 儲存貯體政策。

步驟 4：播放錄製檔

成功設定 CloudFront 發行版並更新儲存貯體政策後，您應該可以使用 IVS 播放程式播放錄製檔：

1. 成功啟動合成，並確定您已將錄製檔存放在 S3 儲存貯體中。
2. 在執行此範例中的步驟 1 到步驟 3 之後，視訊檔案應該可以透過 CloudFront URL 取用。您的 CloudFront URL 是 Amazon CloudFront 主控台中「詳細資料」索引標籤上的分發網域名稱。外觀大致如下：

   ```
   a1b23cdef4ghij.cloudfront.net
   ```

3. 要通過 CloudFront 分發播放錄製的視頻，請在 s3 存儲桶下找到`multivariant.m3u8`文件的對象密鑰。外觀大致如下：

   ```
   FDew6Szq5iTt/9NIpWJHj0wPT/fjFKbylPb3k4/composite/media/hls/
   multivariant.m3u8
   ```

4. 將對象鍵附加到 CloudFront URL 的末尾。您的最終 URL 大致如下：

```
https://a1b23cdef4ghij.cloudfront.net/FDew6Szq5iTt/9NIpWJHj0wPT/
fjFKbylPb3k4/composite/media/hls/multivariant.m3u8
```

5. 現在，您可以將最終 URL 新增到 IVS 播放器的來源屬性中，以觀看完整的錄製檔。若要觀看錄製的影片，您可以使用《IVS 播放器 SDK：Web 指南》當中入門裡的示範。

範例：具有 CloudFront 和 IVS 存取的 S3 儲存貯體政策

下面的程式碼片段說明 S3 儲存貯體政策，該政策 CloudFront 允許將內容讀取到私有儲存貯體和 IVS 以將內容寫入儲存貯體。注意：請勿將下面的程式碼片段複製並貼上到自己的儲存貯體中。您的政策應包含與您的CloudFront 分發和相關的 ID 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"
        }
      }
    }
  ]
}
```

故障診斷

- 組合不會寫入 S3 儲存貯體 — 請確定 S3 儲存貯體和StorageConfiguration 物件已在相同區域中建立。另外，請檢查您的儲存貯體政策，確保 IVS 可以存取值區；請參閱儲存貯體政策。StorageConfiguration
- 表演時我找不到合成 ListCompositions-作品是短暫的資源。一旦變更為最終狀態，就會在幾分鐘後自動刪除。
- 我的合成會自動停止：如果階段上超過 60 秒都沒有發布者，合成將自動停止。

已知問題

在合成進行時，透過複合錄製寫入的媒體播放清單擁有 #EXT-X-PLAYLIST-TYPE:EVENT 標籤。合成完成後，標籤會更新為 #EXT-X-PLAYLIST-TYPE:VOD。為了獲得流暢的播放體驗，我們建議您只在成功完成合成後使用此播放清單。

OBS 和鞭子 Support（實時流媒體）

本文件說明如何使用與 Whip 相容的編碼器 (例如 OBS) 發佈至 IVS 即時串流。鞭（WebRTC-HTTP 攝取協議）是為了標準化 WebRTC 技術攝入而開發的 IETF 草案。

WHIP 實現了與 OBS 等軟件的兼容性，為桌面出版提供了一種替代方案（IVS 廣播 SDK）。熟悉 OBS 的更資深的實況主可能更喜歡它，因為它具有進階生產功能，例如轉換場景，混合音訊和添加圖形浮水印。這為開發人員提供了多功能選擇：使用 IVS 網絡廣播 SDK 進行直接瀏覽器發布，或允許流媒體在其桌面上使用 OBS 以獲得更強大的工具。

此外，在使用 IVS 廣播 SDK 不可行或偏好的情況下，WHIP 也是有益的。例如，在涉及硬體編碼器的設定中，IVS 廣播 SDK 可能不是一個選項。但是，如果編碼器支持 WHIP，您仍然可以直接從編碼器發布到 IVS。

新生物指南

從版本 30 開始，OBS 支持鞭子。首先，請下載 OBS 版本 30 或更新版本：https://obsproject.com/。

若要透過 WHIP 使用 OBS 發佈到 IVS 階段，請依照下列步驟執行：

1. 產生具有發佈功能的參與者權杖。在 WHIP 術語中，參與者令牌是承載令牌。依預設，參與者權杖會在 12 小時後到期，但您可以將持續時間延長至 14 天。
2. 按一下 設定。在「設定」面板的「直播」區段中，從「服務」下拉式清單中選取 WHIP。
3. 對於伺服器，請輸入 https://global.whip.live-video.net/。
4. 對於承載權杖，請輸入您在步驟 2 中產生的參與者權杖。
5. 像平常一樣配置視頻設置，但有一些限制：
   a. IVS 即時串流支援高達 8.5 兆比特的 720p 輸入。如果您超過上述任一限制，您的串流將會中斷連線。
   b. 建議您在「輸出」面板中將「關鍵影格間隔」設定為 1 或 2 秒。較低的關鍵影格間隔可讓觀眾更快速地開始播放影片。我們也建議將「CPU 使用率預設值」設定為「超快」，並將「微調」設定為零延遲，以達到最低的延遲。
   c. 由於 OBS 不支持同步廣播，因此我們建議您將比特率保持在 2.5 Mbps 以下。這可讓使用較低頻寬連線的觀眾觀看。
6. 按開始串流。

Service Quotas (即時串流)

以下是 Amazon Interactive Video Service (IVS) 即時端點、資源和其他操作的服務配額與限制。服務配額 (也稱為限制) 是您的 AWS 帳戶的服務資源或操作數目最大值。也就是說，這些限制以 AWS 帳戶為依據，除非表格中另有說明。另請參閱 AWS Service Quotas。

您可以使用端點，透過程式設計方式連線至 AWS 服務。另請參閱 AWS 服務端點。

所有配額均按區域執行。

Service Quotas 增加

對於可調整的配額，您可以透過 AWS 主控台來請求增加速率。也可以使用主控台檢視服務配額的相關資訊。

API 呼叫速率配額不可調整。

API 呼叫速率配額

端點類型	端點	預設
合成	GetComposition	5 TPS
合成	ListCompositions	5 TPS
合成	StartComposition	5 TPS
合成	StopComposition	5 TPS
MediaEncoder	CreateEncoderConfiguration	5 TPS
MediaEncoder	DeleteEncoderConfiguration	5 TPS
MediaEncoder	GetEncoderConfiguration	5 TPS
MediaEncoder	ListEncoderConfigurations	5 TPS
階段	CreateParticipantToken	50 TPS
階段	CreateStage	5 TPS

端點類型	端點	預設
階段	DeleteStage	5 TPS
階段	DisconnectParticipant	5 TPS
階段	GetParticipant	5 TPS
階段	GetStage	5 TPS
階段	GetStageSession	5 TPS
階段	ListStages	5 TPS
階段	UpdateStage	5 TPS
階段	ListParticipants	5 TPS
階段	ListParticipantEvents	5 TPS
階段	ListStageSessions	5 TPS
StorageConfiguration	CreateStorageConfiguration	5 TPS
StorageConfiguration	DeleteStorageConfiguration	5 TPS
StorageConfiguration	GetStorageConfiguration	5 TPS
StorageConfiguration	ListStorageConfigurations	5 TPS
標籤	ListTagsForResource	10 TPS
標籤	TagResource	10 TPS
標籤	UntagResource	10 TPS

其他配額

資源或功能	預設	可調整	描述
EncoderConfigurations	20	是	每個帳戶編碼器組態資源的最大數量。
合成目的地	2	否	合成資源中目的地物件的最大數量。
合成：最長持續時間	24	否	合成可以存在的時間上限，以小時為單位。
合成	5	是	每個帳戶的最大並行合成資源數。
參與者發布或訂閱持續時間	24	否	參與者可以發布或維持訂閱階段的時間長度上限 (以小時為單位)。
參與者發布解析度	720p	否	參與者發布的影片最大解析度。
參與者下載位元速率	8.5 Mbps	否	所有參與者訂閱的彙總下載位元速率上限。
階段參與者 (發布者)	12	否	一次可以發布到階段的最大參與者人數。
階段參與者 (訂閱用戶)	10,000	是	一次可以訂閱階段的最大參與者人數。
階段	100	是	每個 AWS 區域的階段數目上限。

即時串流優化

為確保您的使用者在使用 IVS 即時串流進行串流和觀看影片時獲得最佳體驗，您可以使用我們目前提供的各種功能，透過多種方式來改善或優化部分體驗。

簡介

針對使用者的體驗品質進行優化時，請務必考慮他們想要的體驗，這些體驗可能視乎使用者觀看的內容和網路狀況而改變。

在本指南中，我們專注於作為串流發布者或串流訂閱用戶的使用者，並且考慮了這些使用者所需的動作和體驗。

適應性串流：使用 Simulcast 進行分層編碼

只有下列用戶端版本才支援此功能：

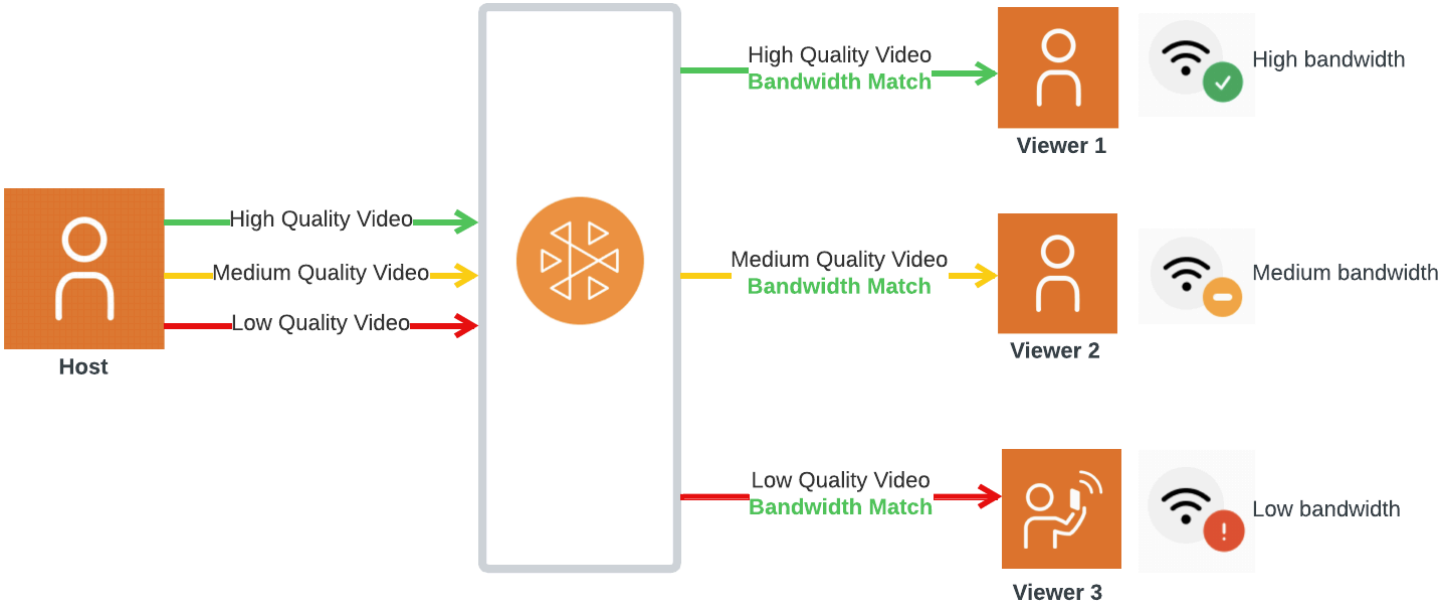
- iOS 和 Android 1.12.0 及更新的版本
- Web 1.5.1 及更新的版本

您必須透amazon-ivs-simulcast過電子郵件傳送 @amazon .com，才能為您的帳戶選擇加入此功能。除非您選擇加入，否則透過 SDK 組態啟用 Simulcast 將不會有任何效果。

只要您選擇使用此功能，在使用 IVS 即時廣播 SDK 時，發布者會對多層影片進行編碼，訂閱用戶則會自動調整或變更為最適合其網路的品質。我們將此稱為使用 Simulcast 進行分層編碼。

Android 和 iOS 以及 Chrome 桌面瀏覽器 (適用於 Windows 和 macOS) 都支援使用 Simulcast 進行分層編碼。我們不支援在其他瀏覽器上進行分層編碼。

在下圖中，主持人將傳送三種影片品質 (高、中和低)。IVS 會根據可用的頻寬，將最高品質的影片轉傳給每位觀眾；這可為每位觀眾提供最佳體驗。如果觀眾 1 的網路連線從良好變更為不佳，IVS 會自動開始向觀眾 1 傳送較低品質的影片，因此觀眾 1 可持續不間斷地觀賞串流 (儘可能保持最佳品質)。



預設層級、品質和影格率

針對行動裝置和 Web 使用者提供的預設品質和層級如下：

行動裝置 (Android、iOS)	Web (Chrome)
高層級 (或自訂)： • 最大位元速率：900,000 bps • 影格率：15 fps • 解析度：360x640	高層級 (或自訂)： • 最大位元速率：1,700,000 bps • 影格率：30 fps • 解析度:1280x720
中層：無 (不需要，因為行動裝置上的高層和低層位元速率之間的差異較小)	中層： • 最大位元速率：700,000 bps • 影格率：20 fps • 解析度：640x360
低層： • 最大位元速率：150,000 bps • 影格率：15 fps • 解析度：180x320	低層： • 最大位元速率：200,000 bps • 影格率：15 fps • 解析度：320x180

設定使用 Simulcast 進行分層編碼

若要在同步轉播中使用分層編碼，您必須選擇加入此功能，並在用戶端上啟用此功能。如果啟用它，您將看到傳輸的整體比特率增加，並具有更少的視頻凍結的好處。

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
```

串流組態

本節將探討您可以對影片和音訊串流做出的其他組態設定。

變更影片串流位元速率

若要變更影片串流的位元速率，請使用下列組態範例。

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
```

變更影片串流影格率

若要變更影片串流的影格率，請使用下列組態範例。

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
```

優化音訊位元速率與立體聲支援

若要變更音訊串流的位元速率與立體聲設定，請使用下列組太範例。

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
```

建議的最佳化

案例	建議
文字串流或移動速度緩慢的內容，例如簡報或幻燈片	採用使用 Simulcast 進行分層編碼，或以較低的影格率設定串流。
具有動作或大量移動的串流	採用使用 Simulcast 進行分層編碼。
具有對話或很少動作的串流	採用使用 Simulcast 進行分層編碼，或選擇純音訊 (請參閱 Web 版、Android 版和 iOS 版即時串流廣播 SDK 指南中的「訂閱參與者」)。
使用者以有限的資料串流	採用使用 Simulcast 進行分層編碼，或者如果您想要降低所有人的資料使用量，請設定較低的影格速率並手動降低位元速率。

資源與支援 (即時串流)

資源

https://ivs.rocks/ 網站專供瀏覽已發佈的內容 (示範、程式碼範例、部落格文章)、估算成本，以及透過即時示範體驗 Amazon IVS。

示範

適用於 iOS 和 Android 的 IVS 即時串流示範向開發人員展示了如何使用 Amazon IVS 建立引人注目的即時社交使用者產生的內容應用程式。此應用程序具有使用者產生的即時串流的可滾動摘要。使用者可以建立影片串流和純音訊房間。影片串流訪客可在訪客位置或對戰 (VS) 模式中加入。如需有關如何部署所需後端和建置應用程式的說明，請參閱下列 GitHub 儲存庫：

- iOS 版：https://github.com/aws-samples/amazon-ivs-real-time-for-ios-demo/
- Android 版：https://github.com/aws-samples/amazon-ivs-real-time-for-android-demo/

- 後端：https://github.com/aws-samples/amazon-ivs-real-time-serverless-demo/

支援

AWS Support 中心提供各種方案，可讓您運用各種工具與專業知識來輔助您的 AWS 解決方案。所有支援方案均提供全年無休的客戶服務。如需技術支援及其他資源來規劃、部署及改善您的 AWS 環境，請選擇最適合您的 AWS 使用案例的支援方案。

AWS Premium Support 是一種一對一的快速回應支援管道，協助您使用 AWS 來建置和執行應用程式。

AWS re:Post 是社群型問答網站，供開發人員討論 Amazon IVS 相關技術問題。

聯絡我們中的連結可用來查詢帳單或帳戶相關問題。對於技術問題，請使用上述開發論壇或支援連結。

詞彙表

另請參閱 AWS 詞彙表。在下表中，LL 代表 IVS 低延遲串流；RT 代表 IVS 即時串流。

術語	描述	LL	RT	聊天
AAC	進階音訊編碼。AAC 是失真數位音訊壓縮的音訊編碼標準。AAC 旨在作為 MP3 格式的後繼者，在相同位元速率下通常比 MP3 實現更高的音效品質。AAC 已被 ISO 和 IEC 標準化，作為 MPEG-2 和 MPEG-4 規格的一部分。	✓	✓	
自適性位元速率串流	自適性位元速率 (ABR) 串流可讓 IVS 播放器在連線品質下降時切換至較低位元速率，並在連線品質改善時切換回較高位元速率。	✓		
自適性串流	請參閱使用聯播進行分層編碼。		✓	
管理使用者	對 AWS 帳戶中可用的資源和服務具有管理存取權的 AWS 使用者。請參閱《AWS 設定使用者指南》中的術語。	✓	✓	✓
ARN	Amazon Resource Name，AWS 資源的唯一識別符。特定 ARN 格式視資源類型而定。如需 IVS 資源使用的 ARN 格式，請參閱服務授權參考。	✓	✓	✓
長寬比	描述影格寬度與影格高度的比率。例如，16:9 是對應於 Full HD 或 1080p 解析度的長寬比。	✓	✓	
音訊模式	針對不同類型的行動裝置使用者及其使用的設備優化的預設或自訂音訊組態。請參閱 IVS 廣播 SDK：行動音訊模式 (即時串流)。		✓	
AVC、H.264、MPEG-4 第 10 部分	進階影片編碼，亦稱為 H.264 或 MPEG-4 第 10 部分，用於失真數位視訊壓縮的影片壓縮標準。	✓	✓	

術語	描述	LL	RT	聊天
背景替換	一種攝影機濾鏡，可讓即時串流創作者變更背景。請參閱 IVS 廣播 SDK：第三方攝影機濾鏡 (即時串流) 中的背景替換。		✓	
位元速率	每秒傳輸或接收位元數的串流指標。	✓	✓	
廣播、廣播者	串流、實況主的其他術語。	✓		
緩衝	當播放裝置在應播放內容之前無法下載內容時發生的情況。緩衝可以透過多種方式建立清單檔案：內容可能會隨機停止和開始 (也稱為卡頓)、內容可能長時間停止 (也稱為凍結)，或 IVS 播放器可能暫停播放。	✓	✓	
位元組範圍播放清單	比標準 HLS 播放清單更精細的播放清單。標準 HLS 播放清單由最多 10 秒的媒體檔案組成。使用位元組範圍播放清單，區段持續時間會與為串流設定的關鍵影格間隔相同。 位元組範圍播放清單僅適用於自動錄製到 S3 儲存貯體的廣播。它是在 HLS 播放清單之外建立的。請參閱自動錄製到 Amazon S3 (低延遲串流) 中的位元組範圍播放清單。	✓		
CBR	固定位元速率，一種編碼器的速率控制方法，可在影片的整個播放過程中保持一致的位元速率，無論廣播期間發生什麼情況。可以填充動作中的間歇以實現所需的位元速率，並且可以透過調整編碼品質以匹配目標位元速率來量化峰值。我們強烈建議您使用 CBR 而非 VBR。	✓	✓	
CDN	內容交付網路或內容分發網路，一種地理位置分散的解決方案，透過使其更接近使用者所在位置來優化串流影片等內容的交付。	✓		

術語	描述	LL	RT	聊天
頻道	儲存串流組態的 IVS 資源，包括擷取伺服器、串流金鑰、播放 URL 和錄製選項。實況主會使用與頻道關聯的串流金鑰來開始廣播。廣播期間產生的所有指標和事件都與頻道資源關聯。	✓		
頻道類型	確定頻道允許的解析度和影格率。請參閱 IVS 低延遲串流 API 參考中的頻道類型。	✓		
聊天記錄	一種進階選項，可以透過將日誌記錄組態與聊天室關聯來加以啟用。			✓
聊天室	一種 IVS 資源，用於儲存聊天工作階段的組態，包括選用功能，例如訊息審查處理常式和聊天記錄。請參閱 IVS 聊天功能入門中的步驟 2：建立聊天室。			✓
用戶端合成	使用主機裝置混合來自階段參與者的音訊和影片串流，然後將這些串流作為複合串流傳送至 IVS 頻道。這樣可以更好地控制合成的外觀，但代價是用戶端資源利用率更高，以及影響觀眾的階段或主持人問題的風險更高。 另請參閱伺服器端合成。	✓	✓	
CloudFront	Amazon 提供的 CDN 服務。	✓		
CloudTrail	一種 AWS 服務，用於收集、監控、分析和保留來自 AWS 與外部來源的事件和帳戶活動。請參閱使用 AWS CloudTrail 記錄 IVS API 呼叫。	✓	✓	✓
CloudWatch	一種 AWS 服務，用於監控應用程式、回應效能變更、優化資源使用，以及深入了解運作狀態。您可以用 CloudWatch 來監視 IVS 指標；請參閱監視 IVS 即時串流和監視 IVS 低延遲串流。	✓	✓	✓
合成	將來自多個來源的音訊和影片串流合併為單一串流的程序。	✓	✓	

術語	描述	LL	RT	聊天
合成管道	合併多個串流並對產生的串流進行編碼所需的一系列處理步驟。	✓	✓	
壓縮	使用比原始表示法更少的位元對資訊進行編碼。任何特定的壓縮都是無失真或失真壓縮。無失真壓縮透過識別和消除統計冗餘來減少位元。無失真壓縮不會遺失任何資訊。失真壓縮透過移除不必要的或不太重要的資訊來減少位元。	✓	✓	
控制平台	儲存有關 IVS 資源的資訊 (例如頻道、階段或聊天室)，並提供建立和管理這些資源的介面。它是區域性的 (基於 AWS 區域)。	✓	✓	✓
CORS	跨來源資源分享，一種 AWS 功能，可讓在一個網域中載入的用戶端 Web 應用程式與不同網域中的 S3 儲存貯體等資源進行互動。您可以根據標頭、HTTP 方法和原始網域設定存取權。請參閱《Amazon Simple Storage Service 使用者指南》中的使用跨來源資源分享 (CORS) – Amazon Simple Storage Service。	✓		
自訂影像來源	IVS 廣播 SDK 提供的介面，可讓應用程式提供自己的影像輸入，而不是僅限於預設攝影機。	✓	✓	
資料平面	將資料從入口傳輸至出口的基礎設施。它根據控制平面中管理的組態運作，且不限於 AWS 區域。	✓	✓	✓
編碼器、編碼	將影片和音訊內容轉換為適合串流的數位格式的程序。編碼可以基於硬體或軟體。	✓	✓	
事件	由 IVS 發佈至 AmazonEventBridge監視服務的自動通知。事件代表串流資源 (例如階段或合成管道) 的狀態或運作狀態變更。請參閱將 Amazon EventBridge 與 IVS 低延遲串流搭配使用，以及將 Amazon EventBridge 與 IVS 即時串流搭配使用。	✓	✓	✓

術語	描述	LL	RT	聊天
FFmpeg	一種免費的開放原始碼軟體專案，由一套用於處理影片和音訊檔案與串流的庫和程式組成。FFmpeg 提供了一個跨平台解決方案來錄製、轉換和串流音訊和影片。	✓		
分段串流	當廣播中斷連接，然後在頻道的錄製組態中指定的時間間隔內重新連接時建立。產生的多個串流視為單一廣播並一起合併到單一錄製的串流。請參閱自動錄製到 Amazon S3 (低延遲串流) 中的合併分段串流。	✓		
影格播放速率	每秒傳輸或接收影片影格數的串流指標。	✓	✓	
HLS	HTTP 即時串流 (HLS)，一種 HTTP 型自適性位元速率串流通訊協定，用於向觀眾交付 IVS 串流。	✓		
HLS 播放清單	組成串流的媒體片段清單。標準 HLS 播放清單由最多 10 秒的媒體檔案組成。HLS 還支援更精細的位元組範圍播放清單。	✓		
主機	將影片和/或音訊傳送至階段的即時事件參與者。		✓	
IAM	Identity and Access Management，一種 AWS 服務，可讓使用者安全地管理身分以及對 AWS 服務和資源 (包括 IVS) 的存取。	✓	✓	✓
擷取	IVS 程序，用於從主持人或廣播者接收影片串流以進行處理或交付給觀眾或其他參與者。	✓	✓	
擷取伺服器	接收影片串流並將其交付給轉碼系統，在該系統中，串流轉碼複用或轉碼為 HLS，以便交付給觀眾。 擷取伺服器是特定的 IVS 元件，用於接收頻道的串流以及擷取協定 (RTMP、RTMPS)。請參閱 IVS 低延遲串流功能入門中有關建立頻道的資訊。		✓	

術語	描述	LL	RT	聊天
交錯影片	僅傳輸和顯示後續影格的奇數行或偶數行，以在不耗用額外頻寬的情況下實現影格率的感知加倍。由於影片品質問題，我們不建議使用交錯影片。	✓	✓	
JSON	JavaScript Object 標記法是一種開放標準檔案格式，它使用人類可讀的文字來傳輸資料物件，其中包含屬性-值配對和陣列資料類型或其他可序列化值。	✓	✓	✓
關鍵影格、差異影格、關鍵影格間隔	關鍵影格 (亦稱為內部編碼或 i 影格) 是影片中影像的全影格。後續影格，差異影格 (亦稱為預測或 p 影格) 僅包含已變更的資訊。關鍵影格將在串流內出現多次，具體取決於編碼器中定義的關鍵影格間隔。	✓	✓	
Lambda	用於執行程式碼 (稱為 Lambda 函數) 而無需佈建任何伺服器基礎設施的 AWS 服務。Lambda 函數可以執行以回應事件和調用請求，或根據排程執行。例如，IVS 聊天功能使用 Lambda 函數為聊天室啟用訊息審查。	✓	✓	✓
延遲、 glass-to-glass延遲	資料傳輸中的延遲。IVS 將延遲範圍定義為： • 低延遲：3 秒以下 • 即時延遲：300 毫秒以下 G 延lass-to-glass遲是指從攝影機擷取即時串流到串流出現在觀眾螢幕上的延遲時間。	✓	✓	
使用聯播進行分層編碼	支援同時編碼和發布具有不同品質等級的多個影片串流。請參閱即時串流優化中的自適性串流：使用聯播進行分層編碼。		✓	
訊息審查處理常式	可讓 IVS 聊天功能客戶在將使用者聊天訊息交付至聊天室之前自動審查/篩選使用者聊天訊息。它是透過將 Lambda 函數與聊天室關聯來啟用的。請參閱聊天訊息審查處理常式中的建立 Lambda 函數。			✓

術語	描述	LL	RT	聊天
混音器	IVS 行動廣播 SDK 的功能，可取得多個音訊和影片來源並產生單一輸出。它支援管理代表來源的螢幕影片和音訊元素，例如攝影機、麥克風、螢幕擷取畫面以及應用程式產生的音訊和影片。然後，輸出可以串流至 IVS。請參閱《IVS 廣播 SDK：混音器指南 (低延遲串流)》中的為混音設定廣播工作階段。	✓		
多主持人串流	將來自多位主持人的串流合併為單一串流。這可以使用用戶端或伺服器端合成來完成。 多主持人串流可以實現邀請觀眾上台問答、主持人比賽、影片聊天、主持人當眾對話等情境。		✓	
多變體播放清單	可用於廣播的所有變體串流的索引。	✓		
OAC	原始存取控制是一種限制 S3 儲存貯體存取的機制，因此只能透過 CloudFrontCDN 提供錄製串流等內容。	✓		
OBS	Open Broadcaster Software，用於影片錄製和即時串流的免費開放原始碼軟體。OBS 為桌面出版提供了一種替代方案 (IVS 廣播 SDK)。熟悉 OBS 的更資深的實況主可能更喜歡它，因為它具有進階生產功能，例如轉換場景，混合音訊和添加圖形浮水印。	✓	✓	
參與者	以主持人或觀眾身分連線至階段的即時使用者。		✓	
參與者權杖	當即時事件參與者加入階段時對其進行身分驗證。參與者字符也可控制參與者是否可以將影片傳送至階段。		✓	

術語	描述	LL	RT	聊天
播放字符、播放金鑰對	一種授權機制，可讓客戶限制私有頻道上的影片播放。播放字符是透過播放金鑰對產生的。 播放金鑰對是公有-私有金鑰對，用來簽署和驗證用於播放的檢視器授權符記。請參閱設定私有頻道中的建立或匯入播放金鑰，並參閱 IVS 低延遲 API 參考中的播放金鑰對端點。	✓		
播放 URL	識別觀眾用來開始播放特定頻道的地址。這個地址可以在全球範圍內使用。IVS 會自動選取 IVS 全球內容交付網路上的最佳位置，以將影片交付給每位觀眾。請參閱 IVS 低延遲串流功能入門中有關建立頻道的資訊。	✓		
私有頻道	可讓客戶使用基於播放字符的授權機制來限制對其串流的存取。請參閱設定私有頻道中的私有頻道的工作流程。	✓		
漸進式影片	依序傳輸和顯示每個影格的所有行。我們建議在廣播的所有階段使用漸進式影片。	✓	✓	
配額	您 AWS 帳戶的 IVS 服務資源或操作數上限。也就是說，這些限制以 AWS 帳戶為依據，除非另有說明。所有配額均按區域執行。請參閱《AWS 一般參考指南》中的 Amazon 互動式影片服務端點和配額。	✓	✓	✓

術語	描述	LL	RT	聊天
區域	可讓您存取實際位於特定地理區域的 AWS 服務。區域提供容錯能力、穩定性和恢復能力，也可降低延遲。透過區域，您可以建立冗餘資源，這些資源會保持可用且不受區域中斷影響。 大多數 AWS 服務請求都與特定地理區域關聯。您在某個區域中建立的資源在任何其他區域中都不存在，除非您明確使用 AWS 服務提供的複寫功能。例如，Amazon S3 支援跨區域複寫。部分服務 (例如，IAM) 沒有跨區域資源。	✓	✓	✓
解析度	描述單一影片影格中的像素數量，例如，Full HD 或 1080p 定義具有 1920x1080 像素的影格。	✓	✓	
根使用者	AWS 帳戶的擁有者。根使用者具有對 AWS 帳戶中所有 AWS 服務和資源的完整存取權。	✓	✓	✓
RTMP、RTMPS	即時訊息協定，透過網路傳輸音訊、影片和資料的業界標準。RTMPS 是 RTMP 的安全版本，透過 Transport Layer Security (TLS/SSL) 連線執行。	✓	✓	
S3 儲存貯體	儲存在 Amazon S3 中的物件集合。許多政策 (包括存取和複寫) 都是在儲存貯體層級定義的，並套用於儲存貯體中的所有物件。例如，IVS 廣播會作為多個物件儲存在 S3 儲存貯體中。	✓		
SDK	軟體開發套件，為使用 IVS 建置應用程式的開發人員提供的程式庫集合。	✓	✓	✓
自拍分割	允許替換即時串流中的背景，使用用戶端特定解決方案，該解決方案接受攝影機影像作為輸入並傳回遮罩，該遮罩為影像的每個像素提供可信度分數，指示它是在前景還是背景。請參閱 IVS 廣播 SDK：第三方攝影機濾鏡 (即時串流) 中的背景替換。		✓	

術語	描述	LL	RT	聊天
語義版本控制	Major.Minor.Patch 形式的版本格式。不影響 API 的錯誤修正會增加修補程式版本，回溯相容的 API 新增/變更會增加次要版本，回溯不相容的 API 變更會增加主要版本。	✓	✓	✓
伺服器端合成	使用 IVS 伺服器來混合階段參與者的音訊和影片，然後將此混合影片傳送至 IVS 頻道，以觸及更多觀眾或將其儲存在 S3 儲存貯體中。伺服器端合成減少了用戶端負載，提高了廣播的恢復能力，並能夠更有效地使用頻寬。 另請參閱用戶端合成。		✓	
Service Quotas	一種 AWS 服務，可協助您從一個位置管理許多 AWS 服務的配額。除了查詢配額值以外，您也可以從 Service Quotas 主控台請求增加配額。	✓	✓	✓
服務連結角色	直接連結至 AWS 服務的獨特類型的 IAM 角色。服務連結角色由 IVS 自動建立，並包含該服務代表您呼叫其他 AWS 服務 (例如存取 S3 儲存貯體) 所需的所有許可。請參閱 IVS 安全性中的使用 IVS 的服務連結角色。	✓		
階段	IVS 資源，代表即時事件參與者可以即時交換影片的虛擬空間。請參閱 IVS 即時串流功能入門中的建立階段。		✓	
階段工作階段	在第一個參與者加入階段時開始，並在最後一位參與者停止發布至階段的幾分鐘後結束。一個長期存放的階段在生命週期內可能有多個工作階段。		✓	
串流	代表從來源連續傳送至目的地的影片或音訊內容的資料。	✓	✓	

術語	描述	LL	RT	聊天
串流金鑰	在您建立頻道時 IVS 指派的識別符；它用於授權串流至頻道。將串流金鑰視為機密，因為具有它的任何人都可以串流至頻道。請參閱 IVS 低延遲串流功能入門。	✓		
串流匱乏	串流交付至 IVS 延遲或停止。當 IVS 未收到編碼裝置公告它在特定時間範圍內會傳送的預期位元量時，會發生這種情況。發生串流匱乏會導致串流匱乏事件。 從觀眾的角度來看，串流匱乏可能會導致影片延遲、緩衝或凍結。串流匱乏可能很短 (少於 5 秒) 或很長 (幾分鐘)，取決於導致串流匱乏的特定情況。請參閱疑難排解常見問答集中的什麼是串流匱乏。	✓	✓	
實況主	將影片或音訊串流傳送至 IVS 的人員或裝置。	✓	✓	
Subscriber	接收主持人的影片和/或音訊的即時事件參與者。請參閱什麼是 IVS 即時串流。		✓	
Tag	您指派給 AWS 資源的中繼資料標籤。標籤可協助您識別和整理 AWS 資源。在 IVS 文件登陸頁面上，請參閱任何 IVS API 文件中的「標記」(用於即時串流、低延遲串流或聊天)。	✓	✓	✓
第三方攝影機濾鏡	可與 IVS 廣播 SDK 整合的軟體元件，允許應用程式在將影像作為自訂影像來源提供給廣播 SDK 之前處理影像。第三方攝影機濾鏡可以處理來自攝影機的影像、套用濾鏡效果等。	✓	✓	
縮圖	從串流中取得的大小縮小的影像。依預設，縮圖每 60 秒產生一次，但可以設定更短的間隔。縮圖解析度取決於頻道類型。請參閱自動錄製到 Amazon S3 (低延遲串流) 中的錄製內容。	✓		

術語	描述	LL	RT	聊天
定時中繼資料	繫結至串流內特定時間戳記的中繼資料。它可以使用 IVS API 以程式設計方式新增，並與特定影格關聯。這可確保所有觀眾都在相對於串流的相同點上接收中繼資料。 定時中繼資料可用於觸發用戶端上的動作，例如在體育賽事期間更新團隊統計資料。請參閱在影片串流中內嵌中繼資料。	✓		
轉碼	將影片和音訊從一種格式轉換為另一種格式。傳入串流可以多個位元率和解析度轉碼為不同的格式，以支援多種播放裝置和網路狀況。	✓	✓	
轉碼複用	將擷取的串流簡單地重新封裝至 IVS，而無需重新編碼影片串流。「轉碼複用」是轉碼多工處理的簡稱，這是一個變更音訊和/或影片檔案格式同時保留部分或全部原始串流的程序。轉碼複用會轉換為不同的容器格式，而不變更檔案內容。與轉碼不同。	✓	✓	
變體串流	相同廣播的一組編碼，具有多個不同的品質等級。每個變體串流都編碼為單獨的 HLS 播放清單。可用變體串流的索引稱為多變體播放清單。 IVS 播放器從 IVS 接收多變體播放清單之後，就可以在播放期間於變體串流之間選擇，並隨著網路條件變更無縫地來回變更。	✓		
VBR	可變位元率，一種編碼器的速率控制方法，它使用在整個播放過程中根據所需的細節層次而變更的動態位元率。由於影片品質問題，我們強烈建議不要使用 VBR；請改用 CBR。	✓	✓	

術語	描述	LL	RT	聊天
檢視	正在主動下載或播放視訊的獨特檢視工作階段。檢視是並行檢視配額的基礎。 當檢視工作階段開始視訊播放時，檢視便會開始。當檢視工作階段停止視訊播放時，檢視結束。播放是觀眾人數的唯一指標；不考慮參與啟發學習法，例如音訊電平、瀏覽器分頁焦點和視訊品質。在計算檢視次數時，IVS 不會考慮個別觀眾的合法性，也不會嘗試對本地化收視人數消除重複，例如在單一機器上的多個影片播放器。請參閱 Service Quotas (低延遲串流) 中的其他配額。	✓		
觀眾	從 IVS 接收串流的人員。	✓		
WebRTC	Web 即時通訊，一個開放原始碼專案，為 Web 瀏覽器和行動應用程式提供即時通訊。它允許通過允許直接通信，從而使音頻和視頻 peer-to-peer 通信在網頁內部工作，無需安裝插件或下載本機應用程序。 WebRTC 技術背後的技術被實現為一個開放的網絡標準，並可作為常規 JavaScript API 在所有主流瀏覽器或作為本機客戶端庫，如 Android 和 iOS。	✓	✓	

術語	描述	LL	RT	聊天
鞭子	WebRTC-HTTP 擷取通訊協定，這是一種以 HTTP 為基礎的通訊協定，可讓 WebRTC 技術將內容擷取到串流服務和/或 CDN 中。鞭是為了標準化 WebRTC 技術攝取而開發的 IETF 草案。 WHIP 實現了與 OBS 等軟件的兼容性，為桌面出版提供了一種替代方案（IVS 廣播 SDK）。熟悉 OBS 的更複雜的實況主可能更喜歡它，因為它具有先進的製作功能，例如場景轉換，音頻混合和疊加圖形 在使用 IVS 廣播 SDK 不可行或偏好的情況下，WHIP 也是有益的。例如，在涉及硬體編碼器的設定中，IVS 廣播 SDK 可能不是一個選項。但是，如果編碼器支持 WHIP，您仍然可以直接從編碼器發布到 IVS。 請參閱 OBS 和鞭子 Support。		✓	
WSS	WebSocket 安全，一種透 WebSockets 過加密 TLS 連線建立的通訊協定。它用於連接至 IVS 聊天功能端點。請參閱 IVS 聊天功能入門中的步驟 4：傳送和接收第一條訊息。			✓

文件歷史記錄 (即時串流)

《即時串流使用者指南》變更

變更	描述	日期
OBS 和鞭子 Support	增加了一個新的頁面。本文件說明如何使用與 Whip 相容的編碼器 (例如 OBS) 發佈至 IVS 即時串流。鞭（WebRTC-HTTP 攝取協議）是為了標準化 WebRTC 技術攝入而開發的 IETF 草案。	2024年2月6日
廣播軟體開發套件:安 iOS 1.14.1, 網頁	在 real-time-streaming 廣播 SDK 指南中更新了新版本的版本號碼和成品鏈接：Android，iOS 和網絡。更新 Amazon IVS 文件登陸頁面上的廣播 SDK 參考文件連結以指向新版本。另請參閱此版本的 Amazon IVS 版本備註。 在安卓手冊中，我們新增了一個新的「已知問題」(影片大小小於 176x176)。 在網路指南中，我們新增了一個新的「已知問題」。解決方法是在調用`getUserMedia` 或時將視頻分辨率限制為 720p。`getDisplayMedia` 在即時串流最佳化中，我們更新了使用同步廣播設定分層編碼；現在預設為停用。	2024年2月1日

廣播軟體開發套件:安卓	在 real-time-streaming 廣播 SDK 指南中更新了新版本的版本號碼和成品鏈接：Android，iOS 和網絡。更新 Amazon IVS 文件登陸頁面上的廣播 SDK 參考文件連結以指向新版本。另請參閱此版本的 Amazon IVS 版本備註。	2024 年 1 月 3 日
IVS 詞彙表	擴展了詞彙表，涵蓋 IVS 即時、低延遲和聊天術語。	2023 年 12 月 20 日
舞台 Health 狀況：新 CloudWatch 量度	將 PacketLoss (階段) 量度重新命名為 DownloadPacketLoss (階段)，並已發行 IVS 即時串流的其他 CloudWatch 量度: • DownloadPacketLoss (舞台、參加者) • DroppedFrames (舞台、參加者) • SubscribeBitrate (舞台, 參加者,MediaType) 請參閱監控 Amazon IVS 即時串流。	2023 年 12 月 7 日
IAM 受管政策	新增兩個受管理的政策：IVS ReadOnlyAccess 和 I FullAccess VS。請參閱： • 安全頁面上有關 Amazon IVS 的受管政策的新章節。 • 變更為 IVS 低延遲串流功能入門中的步驟 3：設定 IAM 許可。	2023 年 12 月 5 日

廣播 SDK：Android 1.13.2 和 iOS 1.13.2	在 real-time-streaming 廣播 SDK 指南中更新了新版本的版本號和成品鏈接：Android 和 iOS。 更新 Amazon IVS 文件登陸頁面上的廣播 SDK 參考文件連結以指向新版本。 另請參閱此版本的 Amazon IVS 版本備註。	2023 年 12 月 4 日
廣播 SDK：Android 1.13.1	在 real-time-streaming 廣播 SDK 指南中更新了新版本的版本號和成品鏈接：Android。 更新 Amazon IVS 文件登陸頁面上的廣播 SDK 參考文件連結以指向新版本。 另請參閱此版本的 Amazon IVS 版本備註。	2023 年 11 月 21 日
Service Quotas	將「參與者發布解析度」從 1080p 變更為 720p。	2023 年 11 月 18 日

廣播 SDK：Android 1.13.0 和 iOS 1.13.0	在 real-time-streaming 廣播 SDK 指南中更新了新版本的版本號和成品鏈接：Android 和 iOS。 更新 Amazon IVS 文件登陸頁面上的廣播 SDK 參考文件連結以指向新版本。 另請參閱此版本的 Amazon IVS 版本備註。 我們也對串流優化進行了各種更新。除此之外，「適應性串流：使用 Simulcast 進行分層編碼」功能現在需要明確選擇加入，並且僅受最新版本的 SDK 支援。	2023 年 11 月 17 日
複合錄製	進行下列變更： • 為此新功能新增了複合錄製頁面。 • 已更新「設定 IAM 許可」政策中的開始使用 IVS 即時串流和 S3 端點。 • 將新端點的呼叫速率配額更新至 Service Quotas。	2023 年 11 月 16 日

伺服器端合成 (SSC)	透過 IVS 伺服器端合成，用戶端可將 IVS 階段的合成和廣播卸載到由 IVS 管理的服務。SSC 和 RTMP 廣播至頻道會透過階段主區域中的 IVS 控制平面端點調用。請參閱： • 開始使用：我們已將 SSC 端點新增至「設定 IAM 許可」中的政策。 • 將 Amazon EventBridge 與 IVS 一起使用 — 我們添加了新的指標。 • 伺服器端合成：此新文件包含概觀與設定指示。 • Service Quotas：我們新增了呼叫頻率限制和其他配額。 另請參閱： • 《IVS 即時串流 API 參考》變更下的變更列表。 • 文件歷史記錄 (低延遲串流) 的變更列表。	2023 年 11 月 16 日
IVS 廣播 SDK	在廣播 SDK 概觀中，我們更新了「平台需求 > 原生平台」來釐清支援的 SDK 版本，並新增了「行動瀏覽器 (iOS 和 Android)」。 在廣播 Web 指南中，我們新增了「行動 Web 限制」。	2023 年 11 月 9 日

IVS 廣播 SDK	我們新增了第三方攝影機濾鏡頁面。	2023 年 11 月 9 日
開始使用 IVS 即時串流	我們更新了設定 IAM 許可的程序。	2023 年 10 月 20 日
監控即時串流	在CloudWatch 量度:IVS 即時串流中，我們新增了維度的範例值。	2023 年 10 月 17 日
廣播 SDK：網路指南	我們對監控遠端參與者媒體靜音狀態進行了幾項變更。	2023 年 10 月 17 日
廣播 SDK：Web 1.6.0	已更新版本的版本號碼和成品連結，請參閱 real-time -streaming 廣播 SDK 指南：Web。 Amazon IVS 文件登陸頁面會指向最新版本的廣播 SDK 參考。 另請參閱此版本的 Amazon IVS 版本備註。 在網路指南的「 MediaStream 從裝置擷取」中，我們也刪除了兩`max`行；最佳做法是僅指定`ideal`。 在即時串流最佳化，我們新了一個新部分，優化音訊位元速率及立體聲支援。	2023 年 10 月 16 日
舞台 Health 狀況：新 CloudWatch 量度	已發佈 IVS 即時串流的 CloudWatch 指標。請參閱監控 Amazon IVS 即時串流。	2023 年 10 月 12 日

廣播 SDK：Android 1.12.1	在 real-time-streaming 廣播 SDK 指南中更新了新版本的版本號和成品鏈接：Android。也新增了使用藍牙麥克風的新章節。 Amazon IVS 文件登陸頁面會指向最新版本的廣播 SDK 參考。 另請參閱此版本的 Amazon IVS 版本備註。	2023 年 10 月 12 日
廣播 SDK：Web 1.5.2	已更新版本的版本號碼和成品連結，請參閱 real-time -streaming 廣播 SDK 指南：Web。 Amazon IVS 文件登陸頁面會指向最新版本的廣播 SDK 參考。 另請參閱此版本的 Amazon IVS 版本備註。	2023 年 9 月 14 日
開始使用 IVS 即時串流	在 Android > 安裝廣播 SDK 中新增了資料繫結。	2023 年 9 月 12 日
廣播 SDK 錯誤處理方式	在下列廣播 SDK 指南中，新增了「錯誤處理」一節：Web、Android 和 iOS。	2023 年 9 月 12 日
開始使用 IVS 即時串流	在分發參與者權杖中，加入了有關不根據目前權杖格式來建置功能的重要備註。	2023 年 9 月 1 日
開始使用 IVS 即時串流	在設定 IAM 許可中，更新了許可集。	2023 年 8 月 31 日

廣播 SDK：Web 1.5.1、Android 1.12.0 和 iOS 1.12.0	更新了新版本的版本號碼和成品連結，請參閱 real-time-streaming 廣播 SDK 指南：網頁版、Android 和 iOS。 更新 Amazon IVS 文件登陸頁面上的廣播 SDK 參考文件連結以指向新版本。 另請參閱此版本的 Amazon IVS 版本備註。	2023 年 8 月 23 日
推出即時串流	此版本隨附重大文件變更。將之前的文件重新命名為 IVS 低延遲串流，並發布了新的 IVS 即時串流文件。IVS 文件登陸頁面現在包含即時串流和低延遲串流的單獨區段。每個區段都有其自己的使用者指南和 API 參考。 如需其他文件變更，請參閱文件歷史記錄 (低延遲串流)。	2023 年 8 月 7 日
廣播 SDK：Web 1.5.0、Android 1.11.0 和 iOS 1.11.0	在 Web、Android 和 iOS 廣播 SDK 指南中更新了新版本的版本編號和成品連結。 更新 Amazon IVS 文件登陸頁面上的廣播 SDK 參考文件連結以指向新版本。 另請參閱此版本的 Amazon IVS 版本備註。	2023 年 8 月 7 日

《IVS 即時串流 API 參考》變更

API 變更	描述	日期
複合錄製	我們新增了 4 個 StorageConfiguration 端點和 7 個物件 (DestinationDetail RecordingConfigurationDestinationConfiguration、、S3、S3、、StorageConfiguration StorageConfiguration、 StorageConfigurationSummary)。 我們修改了 3 個對象（組成，目的地， DestinationConfiguration）。這會影響響 GetComposition 應以及 StartComposition 請求和響應。	2023 年 11 月 16 日
伺服器端合成	我們添加了 8 個構圖和 EncoderConfiguration 端點以及 11 個對象（ChannelDestinationConfiguration構圖 CompositionSummary DestinationConfiguration，目的地 DestinationSummary EncoderConfiguration，EncoderConfigurationSummary， GridConfiguration，LayoutConfiguration，，和視頻）。	2023 年 11 月 16 日
階段運作狀態：新參與者資料	已將六個欄位新增至參與者物件：`browserName` 、`browserVersion` 、`ispName`、`osName`、`osVersion` 及`sdkVersion` 。這會影響響 GetParticipant 應。	2023 年 10 月 12 日
參與者權杖	新增了有關不根據目前權杖格式來建置功能的重要備註。	2023 年 9 月 1 日
推出 IVS 即時串流	此版本隨附重大文件變更。將之前的文件重新命名為 IVS 低延遲串流，並發布了新的 IVS 即時串流文件。IVS 文件登陸頁面現在包含即時串流和低延遲串流的單獨區段。每個區段都有其自己的使用者指南和 API 參考。 IVS 即時串流 API 參考是 IVS 即時串流文件的一部分。之前稱為「IVS 階段 API 參考」。其之前的歷史記錄在文件歷史記錄 (低延遲串流) 中。	2023 年 8 月 7 日

版本備註 (即時串流)

2024年2月6日

OBS 和鞭子 Support

IVS 可與 OBS 等 WIP 相容的編碼器搭配使用，以發佈至 IVS 即時串流。鞭（WebRTC-HTTP 攝取協議）是為了標準化 WebRTC 技術攝入而開發的 IETF 草案。請參閱 OBS 和鞭子 Support 的新頁面。

2024年2月1日

Amazon IVS 廣播開發套件:安 iOS 1.14.1, 網頁 1.8.0 (即時串流)

平台	下載與變更
網路廣播軟體開發套件	參考文檔：https://aws.github.io/ amazon-ivs-web-broadcast /文件/軟件參考 • 現在預設會停用具有同步轉播的分層編碼。 • 修正當「舞台」遭刪除或參與者與伺服器中斷連線時，Stage 實體無法乾淨中斷連線的問題。SDK 現在會發出狀態為 DISCONNECTED （而非然後CONNECTING ）ERRORED 的STAGE_CONNECTION_STATE_CHANGED 事件。 • 修正使用空白音訊或視訊軌道更新策略時，發佈會失敗的問題。
安卓廣播開發套件 1.14.1	參考文檔 amazon-ivs-broadcast-docs：https://aws.github.io/ • 現在預設會停用具有同步轉播的分層編碼。 • libWebRTC 從 M108 更新至 • 修復了幾次崩潰以提高整體穩定性。 • 增加了對立體聲發布的支持。這可以通過 StageAudioConfiguration 對象啟用。

平台	下載與變更
	• 修正了在加入工作階段後，參與者會產生黑色摘要的錯誤。 • 已更新內部libWebRTC 參照，以避免在同一主機應用程式中包含其他libWebRTC 版本時發生符號衝突。
iOS 廣播開發套件 1.14.1	即時串流檔案下載:https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip 參考文件 amazon-ivs-broadcast-docs：https://aws.github.io/ • 現在預設會停用具有同步轉播的分層編碼。 • libWebRTC 從 M108 更新至 • 修復了幾次崩潰以提高整體穩定性。 • 增加了對立體聲發布的支持。這可以透過啟用IVSLocalStageStreamAudioConfiguration 。 • 修復了為其他參與者啟用純音頻模式時崩潰的問題。 • 改進了 TTV 並減少了二進制大小。

廣播 SDK 大小：Android

架構	壓縮大小	未壓縮大小
arm64-v8a	5.223 MB	13.118 MB
armeabi-v7a	4.524 MB	9.134 MB
x86_64	5.418 MB	13.955 MB
x86	5.61 MB	14.369 MB

廣播 SDK 大小：iOS

架構	壓縮大小	未壓縮大小
arm64	3.350 MB	7.790 MB

2024 年 1 月 3 日

Amazon IVS 廣播開發套件:安 iOS 1.13.4, 網頁版 1.7.0 (即時串流)

平台	下載與變更
網路廣播開發套件 1.7.0	參考文檔：https://aws.github.io/ amazon-ivs-web-broadcast /文件/軟件參考 • time-to-video 針對加入階段的訂閱者改善。 • 刪除了`minAudioBitrateKbps` 屬性（它是未使用的）。 • 改善網際網路中斷或變更期間的網路復原功能。
安卓廣播軟體開發套件	參考文檔 amazon-ivs-broadcast-docs：https://aws.github.io/ • StageAudioConfiguration 現在支援設定是否應啟用回音消除功能。
iOS 廣播軟體開發套件	即時串流檔案下載:https://broadcast.live-video.net/1.13.4/AmazonIVSBroadcast-Stages.xcframework.zip 參考文件 amazon-ivs-broadcast-docs：https://aws.github.io/ • 在 iOS 上，我們改進了錄音和播放的音頻引擎，重點放在穩定性和可恢復性。這增強了在

平台	下載與變更
	使用時對路由更改的支持，改善了邊緣機箱的電池回收，並減少主線程阻塞的數量。 • 修正麥克風即使從舞台上分離，仍可能保持作用中狀態的問題，並保持 iOS 隱私權指示器開啟狀態。SDK 當時未處理傳入的音訊。)

廣播 SDK 大小：Android

架構	壓縮大小	未壓縮大小
arm64-v8a	5.187 MB	13.025 MB
armeabi-v7a	4.491 MB	9.056 MB
x86_64	5.359 MB	13.829 MB
x86	5.553 MB	14.214 MB

廣播 SDK 大小：iOS

架構	壓縮大小	未壓縮大小
arm64	3.45 MB	7.84 MB

2023 年 12 月 7 日

新 CloudWatch 量度

我們將 PacketLoss (階段) 量度重新命名為 DownloadPacketLoss (階段)。我們還發布了 IVS 即時串流的其他 CloudWatch 指標：

- DownloadPacketLoss (舞台、參加者)
- DroppedFrames (舞台、參加者)

- SubscribeBitrate (舞台, 參加者,MediaType)

如需詳細資訊，請參閱監控 IVS 即時串流。

2023 年 12 月 4 日

Amazon IVS 廣播 SDK：Android 1.13.2 和 iOS 1.13.2 (即時串流)

平台	下載與變更
所有行動裝置 (Android 和 iOS)	• 開發人員可以啟用/停用雜訊抑制組態以進行發布。
Android 廣播 SDK 1.13.2	參考文檔 amazon-ivs-broadcast-docs：https://aws.github.io/ • 改善了加入工作階段的第一個階時載入影片 (TTV) 所需的時間。
iOS 廣播 SDK 1.13.2	即時串流檔案下載:https://broadcast.live-video.net/1.13.2/AmazonIVSBroadcast-Stages.xcframework.zip 參考文件 amazon-ivs-broadcast-docs：https://aws.github.io/ • 即時 SDK 沒有變更。

廣播 SDK 大小：Android

架構	壓縮大小	未壓縮大小
arm64-v8a	5.177 MB	13.01 MB
armeabi-v7a	4.485 MB	9.045 MB
x86_64	5.352 MB	13.808 MB

架構	壓縮大小	未壓縮大小
x86	5.547 MB	14.192 MB

廣播 SDK 大小：iOS

架構	壓縮大小	未壓縮大小
arm64	3.45 MB	7.82 MB

2023 年 11 月 21 日

Amazon IVS 廣播 SDK：Android 1.13.1 (即時串流)

平台	下載與變更
Android 廣播 SDK 1.13.1	參考文檔 amazon-ivs-broadcast-docs：https://aws.github.io/ • 修正快速離開、釋出和重新加入同一階段時造成當機的問題。

廣播 SDK 大小：Android

架構	壓縮大小	未壓縮大小
arm64-v8a	5.177 MB	13.102 MB
armeabi-v7a	4.485 MB	9.046 MB
x86_64	5.353 MB	13.809 MB
x86	5.547 MB	14.192 MB

2023 年 11 月 17 日

Amazon IVS 廣播 SDK：Android 1.13.0 以及 iOS 1.13.0 (即時串流)

平台	下載與變更
所有行動裝置 (Android 和 iOS)	• 已更新串流優化。除此之外，「適應性串流：使用 Simulcast 進行分層編碼」功能現在需要明確選擇加入，並且僅受最新版本的 SDK 支援。 • 透過減少罕見當機的發生次數，提高階段的穩定性。 • 改善加入階段時載入影片 (TTV) 所需的時間。 • 改進了藍牙裝置的體驗。 • 最佳化 SDK CPU 和記憶體使用率，並減少了程式庫大小。 • 新增了 `StageAudioManager` 類別，可用於設定音訊擷取和播放參數，包括語音通訊、媒體播放等的預設。如需詳細資訊，請參閱新頁面 IVS 廣播 SDK：行動音訊模式。 • 新增了 `requestQualityStats` 函數，可顯示 WebRTC 統計資料的結構化品質事件。 • 新增了函數，可更新音訊位元速率。該函數可像視訊組態一樣，在 `LocalStageStream` 物件上進行設定，但卻透過新的音訊組態物件予以設定。
Android 廣播 SDK 1.13.0	參考文件 amazon-ivs-broadcast-docs：https://aws.github.io/ • `StageRenderer` 介面上的所有方法現在都是選用項目。 • 新增了對 `Surfaceview` 型預覽的支援，可獲得更好的效能。`Session` 和 `StageStream` 中現有的 `getPreview` 方法會繼續傳

平台	下載與變更
	回 TextureView 的子類別，但這可能會在未來的 SDK 版本中發生變更。 • 如果應用程式具體取決於 TextureView，您可以繼續執行而不進行任何變更。您也可以從 getPreview 切換到 getPreviewTextureView，為預設 getPreview 傳回的最終變更做好準備。 • 如果應用程式未明確需要 TextureView，建議您切換為 getPreviewSurfaceView 來降低 CPU 和記憶體使用量。 • SDK 現在實作了一種名為 ImagePreviewSurfaceTarget 的新類型預覽，該預覽可與應用程式提供的 Android Surface 物件搭配使用。這不是 Android View 的子類別，它提供了更好的彈性。 • 修正了在錯誤的時間以錯誤的大小呼叫遠端參與者的 onFrame 回呼的情況。 • SurfaceSource # getInputSurface 現在加上了 @Nullable 註釋。您的代碼應該在使用之前加以檢查。 • 新增了 UserId 和 attributes 至 ParticipantInfo。UserId 和 attributes 屬性內嵌在權杖中，而應用程式可以在參與者加入時透過 ParticipantInfo 擷取它們。 • 攝影機擷取和預覽轉譯現在預設為 720 x 1280 或 15 影格率 (fps) 的發布解析度 (以較高者為準)。您可以使用 StageVideoConfiguration # setCameraCaptureQuality 調整解析度和/或影格率 (fps)。

平台	下載與變更
	• 設定組態屬性時拋出的 `IllegalArgumentException` 現在在例外狀況訊息中包含提供的值。
iOS 廣播 SDK 1.13.0	即時串流檔案下載:https://broadcast.live-video.net/1.13.0/AmazonIVSBroadcast-Stages.xcframework.zip 參考文件 amazon-ivs-broadcast-docs：https://aws.github.io/ • 修正了如果在發布前更新視訊組態，SDK 不會變更視訊組態的問題。 • 納入了針對 LibVPX 安全漏洞 (CVE-2023-5217) 的 Google 修正程式。(請注意，Android SDK 不需要對此問題進行任何更改。) • 使用包含 `libWebRTC` 的其他程式庫的應用程式不會再與 IVS 廣播 SDK 發生衝突。 • `IVSStageRenderer` 通訊協定上的所有方法現在都會標記為 `@optional` 。 • 我們的 SDK 傳回的麥克風和攝影機現在都會有保證的排列順序，如 SDK 本身所述。 • 現在，多部攝影機的 `isDefault` 屬性值可為 `true`，由作業系統決定的每個位置各一個。 • 新增了 `IVSStageAudioManager` ，可以精確控制基礎 `AVAudioSession` ，為階段功能提供更廣泛的使用案例。 • 已新增 `UserId` 到 `ParticipantInfo` 。

廣播 SDK 大小：Android

架構	壓縮大小	未壓縮大小
arm64-v8a	5.17 MB	13.00 MB
armeabi-v7a	4.48 MB	9.04 MB
x86_64	5.35 MB	13.80 MB
x86	5.54 MB	14.18 MB

廣播 SDK 大小：iOS

架構	壓縮大小	未壓縮大小
arm64	3.45 MB	7.84 MB

2023 年 11 月 16 日

複合錄製

此新功能可將 IVS 階段的複合檢視錄製到 Amazon S3 儲存貯體。如需詳細資訊，請參閱：

- 複合錄製：這是一個新頁面。
- 開始使用 IVS 即時串流：我們在「設定 IAM 許可」的政策中新增了 S3 端點。
- Service Quotas：我們已新增新端點的呼叫速率配額。
- IVS 即時串流 API 參考 — 我們新增了 4 個 StorageConfiguration 端點和 7 個物件 (DestinationDetail、 RecordingConfiguration、S3 DestinationConfiguration、S3Detail、S3 StorageConfiguration、、、)。 StorageConfiguration StorageConfigurationSummary我們還修改了 3 個對象（組成，目的地， DestinationConfiguration）；這會影響響 GetComposition 應以及 StartComposition 請求和響應。

2023 年 11 月 16 日

伺服器端合成

透過 IVS 伺服器端合成，用戶端可將 IVS 階段的合成和廣播卸載到由 IVS 管理的服務。伺服器端合成和 RTMP 廣播至頻道會透過階段主區域中的 IVS 控制平面端點調用。如需詳細資訊，請參閱：

- 開始使用 IVS 即時串流：我們在「設定 IAM 許可」的政策中新增了 SSC 端點。
- 將 Amazon EventBridge 與 IVS 即時串流搭配使用 — 我們新增了新的指標。
- 伺服器端合成：此新文件包含概觀與設定指示。
- Service Quotas (即時串流)：我們新增了呼叫頻率限制和其他配額。
- 即時串流 API 參考 — 我們新增了 8 個構成和 EncoderConfiguration端點以及 11 個物件 (構成ChannelDestinationConfiguration CompositionSummary DestinationConfiguration、目的地 DestinationSummary、 EncoderConfiguration、 EncoderConfigurationSummary、、GridConfiguration、 LayoutConfiguration、、和視訊)。

在《IVS 低延遲串流使用者指南》中，參閱：

- 在 IVS 串流上啟用多位主持人：我們新增了「廣播階段：用戶端與伺服器端合成」，並更新了「4. 廣播階段。」

2023 年 10 月 16 日

Amazon IVS 廣播 SDK：Web 1.6.0 (即時串流)

平台	下載與變更
Web 廣播 SDK 1.6.0	參考文檔：https://aws.github.io/ amazon-ivs-web-broadcast /文件/軟件參考 • 改善影片播放時間 (TTV)。 • 增加了 `maxAudioBitrate` 設定，支援高達 128kbps 的單聲道或立體聲音訊通道。

2023 年 10 月 12 日

新 CloudWatch 指標和參與者資料

我們發布了 IVS 即時串流的 CloudWatch 指標。如需詳細資訊，請參閱監控 IVS 即時串流。

我們也在參與者 API 物件中新增六個欄位：`browserName`、`browserVersion`、`ispName`、`osName`、`osVersion` 及 `sdkVersion`。這會影響響 GetParticipant 應。請參閱 IVS 即時串流 API 參考。

2023 年 10 月 12 日

Amazon IVS 廣播 SDK：Android 1.12.1 (即時串流)

平台	下載與變更
Android 廣播 SDK 1.12.1	參考文件 amazon-ivs-broadcast-docs：https://aws.github.io/ • 修正了呼叫 `BroadcastSession.setListener` 導致錯誤的錯誤。

廣播 SDK 大小：Android

架構	壓縮大小	未壓縮大小
arm64-v8a	5.853 MB	16.375 MB
armeabi-v7a	4.895 MB	10.803 MB
x86_64	6.149 MB	17.318 MB
x86	6.328 MB	17.186 MB

2023 年 9 月 14 日

Amazon IVS 廣播 SDK：Web 1.5.2 (即時串流)

平台	下載與變更
Web 廣播 SDK 1.5.2	參考文檔：https://aws.github.io/ amazon-ivs-web-broadcast /文件/軟件參考 • 修正當已發布的狀態進入 ERRORED 狀態時，無法以 `refreshStrategy` 重新發布的錯誤。

2023 年 8 月 23 日

Amazon IVS 廣播 SDK：Web 1.5.1、Android 1.12.0，以及 iOS 1.12.0 (即時串流)

平台	下載與變更
Web 廣播 SDK 1.5.1	參考文檔：https://aws.github.io/ amazon-ivs-web-broadcast /文件/軟件參考 • 修正了 TypeScript 5 上內部可能類型的錯誤。 • 為 Simulcast 支援加入了更好的偵測能力。 • 修正了嘗試發布時，兩個具有 `refreshStrategy` 的競爭條件。 • 修正了嘗試更新要訂閱的參與者時，一個具有 `refreshStrategy` 的競爭條件。
所有行動裝置 (Android 和 iOS)	• 修正了發布動作永遠不會完成的罕見問題。 • 透過減少罕見當機的發生次數，提高階段的穩定性。 • 透過解決快速加入 / 離開造成的競爭條件問題，改善階段的穩定性。

平台	下載與變更
	• 在 `ImageDevice` 上加入了新的 `setOnFrameCallback` 方法。如此可在影格通過裝置本身時進行觀察，深入瞭解最新影像的長寬比。此方法也可用來偵測為階段中的遠端參與者轉譯第一個影格的時機。
Android 廣播 SDK 1.12.0	參考文件 amazon-ivs-broadcast-docs：https://aws.github.io/ • 現在已支援 Android 9。 • 改善 CPU 使用率和效能。
iOS 廣播 SDK 1.12.0	即時串流檔案下載:https://broadcast.live-video.net/1.12.0/AmazonIVSBroadcast-Stages.xcframework.zip 參考文件 amazon-ivs-broadcast-docs：https://aws.github.io/ • 更正了 `IVSDeviceDiscovery.createAudioSourceWithName` 的簽名以傳回 `IVSCustomAudioSource` 而不是 `IVSCustomImageSource` 。

廣播 SDK 大小：Android

架構	壓縮大小	未壓縮大小
arm64-v8a	5.853 MB	16.375 MB
armeabi-v7a	4.895 MB	10.803 MB
x86_64	6.149 MB	17.318 MB
x86	6.328 MB	17.186 MB

廣播 SDK 大小：iOS

架構	壓縮大小	未壓縮大小
arm64	5.06 MB	10.92 MB

2023 年 8 月 7 日

Amazon IVS 廣播 SDK：Web 1.5.0、Android 1.11.0 和 iOS 1.11.0

平台	下載與變更
Web 廣播 SDK 1.5.0	參考文檔：https://aws.github.io/ amazon-ivs-web-broadcast /文件/軟件參考 • 新增了 Simulcast – 啟用時，此功能允許發布者傳送高品質和低品質的影片層。訂閱用戶會根據網路狀況自動選取最佳品質。請參閱優化媒體。
所有行動裝置 (Android 和 iOS)	新增了 Simulcast – 啟用時，此功能允許發布者傳送高品質和低品質的影片層。訂閱用戶會根據網路狀況自動選取最佳品質。請參閱 Android 版和 iOS 版廣播 SDK 指南中的「啟用/停用使用 Simulcast 進行分層編碼」。
Android 廣播 SDK 1.11.0	參考文件 amazon-ivs-broadcast-docs：https://aws.github.io/ • 修正了建立許多階段最終導致當機的問題。(階段的確切數目取決於裝置。)
iOS 廣播 SDK 1.11.0	適用於即時串流的下載：https://broadcast.live-video.net/1.11.0/AmazonIVSBroadcast-Stages.xcframework.zip

平台	下載與變更
	參考文件 amazon-ivs-broadcast-docs：https://aws.github.io/ • 更正了 `IVSDeviceDiscovery.createAudioSourceWithName` 的簽名以傳回 `IVSCustomAudioSource` 而不是 `IVSCustomImageSource` 。

廣播 SDK 大小：Android

架構	壓縮大小	未壓縮大小
arm64-v8a	5.811 MB	16.186 MB
armeabi-v7a	4.857 MB	10.646 MB
x86_64	6.108 MB	17.122 MB
x86	6.289 MB	16.994 MB

廣播 SDK 大小：iOS

架構	壓縮大小	未壓縮大小
arm64	5.030 MB	10.810 MB

2023 年 8 月 7 日

即時串流

藉由 Amazon Interactive Video Service (IVS) 即時串流，您能夠以從主持人到觀眾不到 300 毫秒的延遲交付即時串流。

此版本隨附重大文件變更。IVS 文件登陸頁面現在包含即時串流和低延遲串流的單獨區段。每個區段都有其自己的使用者指南和 API 參考。如需文件詳細資訊，請參閱文件歷史記錄 (即時和低延遲文件變更)。如需即時串流相關資訊，請先參閱 IVS Real-Time Streaming User Guide 和 IVS Real-Time Streaming API Reference。

本文為英文版的機器翻譯版本，如內容有任何歧義或不一致之處，概以英文版為準。