



即時串流使用者指南

Amazon IVS



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，這些擁有者可能附屬於 Amazon，或與 Amazon 有合作關係，亦或受到 Amazon 贊助。

Table of Contents

什麼是 IVS 即時串流？	1
全球解決方案、區域控制	1
串流和檢視是全球性的	1
控制是區域性的	2
IVS 入門	3
簡介	3
必要條件	3
其他參考	3
即時串流術語	4
步驟概觀	4
步驟 1：設定 IAM 許可	5
對 IVS 許可使用現有的政策	5
選用：為 Amazon IVS 許可建立自訂政策	5
建立新使用者並新增許可	7
為現有的使用者新增許可	8
步驟 2：建立階段	8
主控台說明	8
CLI 說明	9
步驟 3：分發參與者權杖	10
使用金鑰對建立權杖	10
使用 IVS 即時串流 API 建立權杖	15
步驟 4：整合 IVS 廣播 SDK	17
Web	17
Android	18
iOS	19
步驟 5：發布和訂閱影片	20
IVS 主控台	21
Web	21
Android	29
iOS	53
監控	83
什麼是階段工作階段？	83
檢視階段工作階段和參與者	83
主控台說明	83

檢視參與者的事件	83
主控台說明	83
CLI 說明	84
存取 CloudWatch 指標	85
CloudWatch 主控台說明	85
CLI 說明	85
CloudWatch 指標：IVS 即時串流	86
IVS 廣播 SDK	90
平台需求：	90
原生平台	90
桌面瀏覽器	91
移動瀏覽器 (iOS 和 Android)	91
Webview	92
必要的裝置存取權	92
支援	92
版本控制	92
Web 指南	93
開始使用	93
發布與訂閱	96
已知問題和解決方法	107
錯誤處理	109
Android 指南	111
開始使用	112
發布與訂閱	115
已知問題和解決方法	125
錯誤處理	126
iOS 指南	129
開始使用	129
發布與訂閱	131
iOS 如何選擇攝影機解析度和影格速率	139
已知問題和解決方法	141
錯誤處理	142
自訂圖像來源	144
Android	144
iOS	145
第三方攝影機濾鏡	145

整合第三方攝影機濾鏡	146
BytePlus	146
DeepAR	148
Snap	148
背景替換	163
行動音訊模式	183
簡介	183
音訊模式預設值	184
進階使用案例	186
與其他 SDK 整合	188
搭配 IVS 使用 Amazon EventBridge	190
為 Amazon IVS 建立 Amazon EventBridge 規則	191
範例：合成狀態變更	191
範例：個別參與者錄製狀態變更	195
範例：階段更新	197
伺服器端合成	199
概觀	199
優勢	200
合成生命週期	201
IVS API	202
版面配置	202
開始使用	204
必要條件	205
CLI 說明	205
啟用螢幕共用	208
建立 EncoderConfiguration 資源	208
啟動合成	208
停止合成	210
錄製	212
個別參與者錄製	212
複合錄製	212
個別參與者錄製	213
簡介	213
工作流程	214
純音訊錄製	217
錄製內容	217

JSON 中繼資料檔案	218
複合錄製	222
.....	212
錄製內容	225
StorageConfiguration 的儲存貯體政策	226
JSON 中繼資料檔案	226
從私人儲存貯體播放錄製的內容	230
故障診斷	234
已知問題	235
串流擷取	236
支援的通訊協定	236
支援的媒體規格	237
RTMP	237
建立舞台	237
建立擷取組態	238
使用 RTMP 編碼器發布	238
WHIP	239
OBS 指南	239
Service Quotas	241
Service Quotas 增加	241
API 呼叫速率配額	241
.....	241
其他配額	243
.....	243
串流優化	245
簡介	245
適應性串流：使用 Simulcast 進行分層編碼	245
預設層級、品質和影格率	246
分層解析度	246
設定 Simulcast 分層編碼	247
串流組態	248
變更影片串流位元速率	248
變更影片串流影格率	249
優化音訊位元速率與立體聲支援	250
變更訂閱用戶抖動緩衝區最低延遲	251
建議的最佳化	253

資源與支援	254
資源	254
示範	254
支援	255
詞彙表	256
文件歷史記錄	270
《即時串流使用者指南》變更	270
《IVS 即時串流 API 參考》變更	285
版本備註	288
2024 年 10 月 10 日	288
IVS 廣播 SDK : Web 1.17.0 (即時串流)	288
2024 年 10 月 10 日	288
Amazon IVS 廣播 SDK : Android 1.23.0、iOS 1.23.0 (即時串流)	288
2024 年 9 月 11 日	289
Amazon IVS 廣播 SDK : Android 1.22.0、iOS 1.22.0 (即時串流)	289
2024 年 9 月 11 日	291
IVS 廣播 SDK : Web 1.16.0 (即時串流)	291
2024 年 9 月 9 日	291
RTMP 擷取	291
2024 年 8 月 19 日	291
主控台內發布/訂閱	291
2024 年 8 月 15 日	291
IVS 廣播 SDK : Web 1.15.0 (即時串流)	291
2024 年 8 月 15 日	292
Amazon IVS 廣播 SDK : Android 1.21.0、iOS 1.21.0 (即時串流)	292
2024 年 7 月 18 日	294
IVS 廣播 SDK : Web 1.14.0 (即時串流)	294
2024 年 7 月 18 日	294
Amazon IVS 廣播 SDK : Android 1.20.0、iOS 1.20.0 (即時串流)	294
2024 年 6 月 26 日	295
使用金鑰對產生參與者權杖	295
2024 年 6 月 20 日	295
個別參與者錄製	295
2024 年 6 月 13 日	296
Amazon IVS 廣播 SDK : Android 1.19.0、iOS 1.19.0 (即時串流)	296
2024 年 6 月 13 日	297

IVS 廣播 SDK : Web 1.13.0 (即時串流)	297
2024 年 5 月 20 日	298
IVS 廣播 SDK : Web 1.12.0 (即時串流)	298
2024 年 5 月 16 日	298
Amazon IVS 廣播 SDK : Android 1.18.0、iOS 1.18.0 (即時串流)	298
2024 年 5 月 6 日	299
IVS 廣播 SDK : Web 1.11.0 (即時串流)	299
2024 年 4 月 30 日	300
IVS 廣播 SDK : Web 1.10.1 (即時串流)	300
2024 年 4 月 30 日	300
Amazon IVS 廣播 SDK : Android 1.15.2、iOS 1.15.2 (即時串流)	300
2024 年 4 月 22 日	301
Amazon IVS 廣播 SDK : Android 1.17.0、iOS 1.17.0 (即時串流)	301
2024 年 3 月 21 日	303
Amazon IVS 廣播 SDK : Android 1.16.0、iOS 1.16.0、Web 1.10.0 (即時串流)	303
2024 年 3 月 13 日	304
Amazon IVS 廣播 SDK : Android 1.15.1、iOS 1.15.1 (即時串流)	304
2024 年 3 月 13 日	305
伺服器端合成 API 更新	305
2024 年 3 月 8 日	305
伺服器端合成版面配置更新	305
2024 年 2 月 22 日	306
Amazon IVS 廣播 SDK : Android 1.15.0、iOS 1.15.0、Web 1.9.0 (即時串流)	306
2024 年 2 月 7 日	307
伺服器端合成版面配置更新	307
2024 年 2 月 6 日	309
OBS 和 WHIP 支援	309
2024 年 2 月 1 日	310
Amazon IVS 廣播 SDK : Android 1.14.1、iOS 1.14.1、Web 1.8.0 (即時串流)	310
2024 年 1 月 3 日	312
Amazon IVS 廣播 SDK : Android 1.13.4、iOS 1.13.4、Web 1.7.0 (即時串流)	312
2023 年 12 月 7 日	313
新的 CloudWatch 指標	313
2023 年 12 月 4 日	314
Amazon IVS 廣播 SDK : Android 1.13.2 和 iOS 1.13.2 (即時串流)	314
2023 年 11 月 21 日	315

Amazon IVS 廣播 SDK : Android 1.13.1 (即時串流)	315
2023 年 11 月 17 日	316
Amazon IVS 廣播 SDK : Android 1.13.0 以及 iOS 1.13.0 (即時串流)	316
2023 年 11 月 16 日	319
複合錄製	319
2023 年 11 月 16 日	320
伺服器端合成	320
2023 年 10 月 16 日	320
Amazon IVS 廣播 SDK : Web 1.6.0 (即時串流)	320
2023 年 10 月 12 日	321
新的 CloudWatch 指標和參與者資料	321
2023 年 10 月 12 日	321
Amazon IVS 廣播 SDK : Android 1.12.1 (即時串流)	321
2023 年 9 月 14 日	322
Amazon IVS 廣播 SDK : Web 1.5.2 (即時串流)	322
2023 年 8 月 23 日	322
Amazon IVS 廣播 SDK : Web 1.5.1、Android 1.12.0 , 以及 iOS 1.12.0 (即時串流)	322
2023 年 8 月 7 日	324
Amazon IVS 廣播 SDK : Web 1.5.0、Android 1.11.0 和 iOS 1.11.0	324
2023 年 8 月 7 日	326
即時串流	326

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 25,000 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布的內容 (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： <ul style="list-style-type: none">將 <code>--region <aws-region></code> 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 即時串流功能整合到應用程式所涉及的步驟。

主題

- [IVS 即時串流簡介](#)
- [步驟 1：設定 IAM 許可](#)
- [步驟 2：建立階段](#)
- [步驟 3：分發參與者權杖](#)
- [步驟 4：整合 IVS 廣播 SDK](#)
- [步驟 5：發布和訂閱影片](#)

IVS 即時串流簡介

本節列出使用即時串流的先決條件，並介紹關鍵術語。

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. [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](#) – 將影片傳送至舞台，並從其他主機接收影片：[IVS 主控台](#)、[the section called “Web”](#)、[the section called “Android”](#) 和 [the section called “iOS”](#)。

步驟 1：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。

步驟 2：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 端點來建立階段。

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。

Amazon IVS > Video > Stages > Create stage

Create stage [Info](#)

A stage allows participants to send and receive video and audio with others in real time. You can broadcast a stage to a channel, allowing viewers to see and hear stage participants without needing to join the stage directly. [Learn more](#) [↗](#)

▶ How Amazon IVS stages work

Setup

Stage name – *optional*

Maximum length: 128 characters. May include numbers, letters, underscores (_) and hyphens (-).

▶ Tags [Info](#)

A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

Cancel

Create stage

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[安裝或更新至最新版 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/VSWjvX5X0kU3",  
  "name": "test-stage"  
}  
}
```

步驟 3：分發參與者權杖

現在您有了一個舞台，需要建立權杖，並將其分發給參與者，使參與者能夠加入此舞台，並開始傳送和接收影片。有兩種權杖產生方法：

- 使用金鑰對[建立](#)權杖。
- [使用 IVS 即時串流 API 建立](#)權杖。

這兩種方法皆在下面有所描述。

使用金鑰對建立權杖

您可以在伺服器應用程式建立權杖，然後將權杖分發給參與者，用以加入舞台。您需要產生 ECDSA 公有/私有金鑰對來簽署 JWT，並且將公有金鑰匯入至 IVS。然後，IVS 就可以在有人加入舞台時驗證權杖。

IVS 不會自動刪除到期的金鑰。如果私有金鑰遭到盜用，您必須刪除舊的公有金鑰。

建立新的金鑰對

有多種方法可以建立金鑰對。以下提供兩個範例。

若要在主控台中建立新的金鑰對，請依照下列步驟執行：

1. 開啟 [Amazon IVS 主控台](#)。如果您尚未在舞台中，請選擇舞台區域。
2. 在左側導覽功能表中，選擇即時串流 > 公有金鑰。
3. 選擇建立公有金鑰。建立公有金鑰對話方塊隨即出現。
4. 依照提示進行操作並選擇 Create (建立)。
5. Amazon IVS 會產生新的金鑰對。公有金鑰會作為公有金鑰資源匯入，而私有金鑰可立即用於下載。如有需要，也可以稍後下載公有金鑰。

Amazon IVS 會在用戶端產生金鑰，而不會儲存私有金鑰。請務必儲存金鑰；稍後無法擷取金鑰。

若要使用 OpenSSL 建立新 P384 EC 金鑰對 (您可能必須先安裝 [OpenSSL](#))，請依照下列步驟執行。此過程可讓您存取私有金鑰和公有金鑰。只有當您想測試符記的驗證時，才需要公有金鑰。

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

現在，按照下列指示匯入新的公有金鑰。

匯入公有金鑰

在您取得金鑰對後，就可以將公有金鑰匯入 IVS。我們的系統不需要私有金鑰，但您可以用來簽署符記。

若要使用主控台匯入現有的公有金鑰：

1. 開啟 [Amazon IVS 主控台](#)。如果您尚未在舞台中，請選擇舞台區域。
2. 在左側導覽功能表中，選擇即時串流 > 公有金鑰。
3. 選擇匯入。匯入公有金鑰對話方塊隨即出現。
4. 依照提示進行操作並選擇 Import (匯入)。
5. Amazon IVS 會匯入公有金鑰並產生公有金鑰資源。

若要使用 CLI 匯入現有的公有金鑰：

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

如果區域位於本機 AWS 組態檔案中，您可以省略 `--region <aws-region>`。

這是回應範例：

```
{
  "publicKey": {
    "arn": "arn:aws:ivs:us-west-2:123456789012:public-key/f99cde61-c2b0-4df3-8941-
ca7d38acca1a",
    "fingerprint": "98:0d:1a:a0:19:96:1e:ea:0a:0a:2c:9a:42:19:2b:e7",
    "publicKeyMaterial": "-----BEGIN PUBLIC KEY-----
\nMHYwEAYHKoZIzj0CAQYFK4EEACIDYgAEVjYMV+P4ML6xemanCrtse/FDwsNnpYmS
\nS6vRV9Wx37mjwi02h0bKuCJqpj7x0lpz0bHm5v1JBvdZYAd/r2LR5aChK+/GM2Wj
\nl8MG9NJIVFaw1u3bvjEjzTASSfS1BDX1\n-----END PUBLIC KEY-----\n",
    "tags": {}
  }
}
```



```
}  
}
```

API 請求

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

產生和簽署權杖

如需有關使用 JWT 以及用於簽章符記的支援程式庫的詳細資訊，請造訪 jwt.io。在 jwt.io 介面上，您必須輸入私有金鑰才能簽署符記。只有在您想驗證符記時才需要公有金鑰。

所有 JWT 都有三個欄位：標頭、承載和簽章。

JWT 標頭和承載的 JSON 結構描述如下所述。您也可以從 IVS 主控台複製範例 JSON。若要從 IVS 主控台取得標頭和承載 JSON：

1. 開啟 [Amazon IVS 主控台](#)。如果您尚未在舞台中，請選擇舞台區域。
2. 在左側導覽功能表中，選擇即時串流 > 舞台。
3. 選取您要使用的舞台。請選取檢視詳細資訊。
4. 在參與者權杖區段中，選取建立權杖旁的下拉式清單。
5. 選取建置權杖標頭和承載。
6. 填寫表單並複製彈出視窗底部顯示的 JWT 標頭和承載。

權杖結構描述：標頭

標頭指定：

- alg 是簽章演算法。這是 ES384，一種使用 SHA-384 雜湊演算法的 ECDSA 簽章演算法。
- typ 是符記類型，JWT。
- kid 是用來簽署權杖的公有金鑰 ARN。此 ARN 必須與從 [GetPublicKey](#) API 請求傳回的 ARN 相同。

```
{
```

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

權杖結構描述：承載

此承載包含 IVS 特定資料。除了 `user_id` 之外的所有欄位都是必填欄位。

- JWT 規格中的 `RegisteredClaims` 是保留宣告，必須提供此宣告，舞台權杖才會有效：
 - `exp` (過期時間) 是權杖過期時間的 Unix UTC 時間戳記。(Unix 時間戳記是一個數值，表示從 1970-01-01T00:00:00Z UTC 到指定 UTC 日期/時間的秒數 (忽略閏秒)。) 參與者加入舞台時，系統會驗證此權杖。IVS 提供預設 12 小時 (我們的建議值) TTL 的權杖；此時間最多可延長至發行時間 (`iat`) 後 14 天。此值必須為整數類型。
 - `iat` (發行時間) 是 JWT 發行時間的 Unix UTC 時間戳記。(請參閱有關 Unix 時間戳記 `exp` 的備註。) 此值必須為整數類型。
 - `jti` (JWT ID) 是參與者 ID，此 ID 會用於追蹤和指代獲得權杖的參與者。每個權杖都必須具有唯一的參與者 ID。此 ID 必須是區分大小寫的字串、長度上限為 64 個字元，且只包含英數字元、連字號 (-) 和底線 (_) 字元。不允許使用其他特殊字元。
- `user_id` 是客戶指派名稱 (選用)，可協助識別權杖，可用來將參與者連結至客戶自有系統中的使用者。它應與 [CreateParticipantToken](#) API 請求中的 `userId` 欄位相符。它可以是任何 UTF-8 編碼文字，而且是最多 128 個字元的字串。此欄位會向所有舞台參與者顯示，而此資訊不應用於個人身分識別、機密或敏感資訊。
- `resource` 是舞台的 ARN；例如 `arn:aws:ivs:us-east-1:123456789012:stage/oRmLNwuCeMlQ`。
- `topic` 是舞台的 ID，可以從舞台 ARN 中擷取。例如，如果舞台 ARN 為 `arn:aws:ivs:us-east-1:123456789012:stage/oRmLNwuCeMlQ`，則舞台 ID 為 `oRmLNwuCeMlQ`。
- `events_url` 必須是 `CreateStage` 或 `GetStage` 操作傳回的事件端點。建議您在建立舞台時快取此值，可對該值進行快取的時間最長為 14 天。範例值為 `wss://global.events.live-video.net`。
- `whip_url` 必須是 `CreateStage` 或 `GetStage` 操作傳回的 WHIP 端點。建議您在建立舞台時快取此值，可對該值進行快取的時間最長為 14 天。範例值為 `https://453fdfd2ad24df.global-bm.whip.live-video.net`。
- `capabilities` 會指定權杖的功能；有效值為 `allow_publish` 和 `allow_subscribe`。對於僅限訂閱的權杖，請僅將 `allow_subscribe` 設定為 `true`。

- `attributes` 是選填欄位，您可以在其中指定應用程式提供的屬性，以編碼至權杖並連接至舞台。映射的鍵值可以包含 UTF-8 編碼文字。此欄位的長度上限總計為 1 KB。此欄位會向所有舞台參與者顯示，而此資訊不應用於個人身分識別、機密或敏感資訊。
- `version` 必須為 `1.0`。

```
{
  "exp": 1697322063,
  "iat": 1697149263,
  "jti": "Mx6clRRHODPy",
  "user_id": "<optional_customer_assigned_name>",
  "resource": "<stage_arn>",
  "topic": "<stage_id>",
  "events_url": "wss://global.events.live-video.net",
  "whip_url": "https://114ddfabadaf.global-bm.whip.live-video.net",
  "capabilities": {
    "allow_publish": true,
    "allow_subscribe": true
  },
  "attributes": {
    "optional_field_1": "abcd1234",
    "optional_field_2": "false"
  },
  "version": "1.0"
}
```

權杖結構描述：簽章

若要建立簽章，請使用私有金鑰搭配標頭 (ES384) 中指定的演算法來簽署已編碼的標頭和承載。

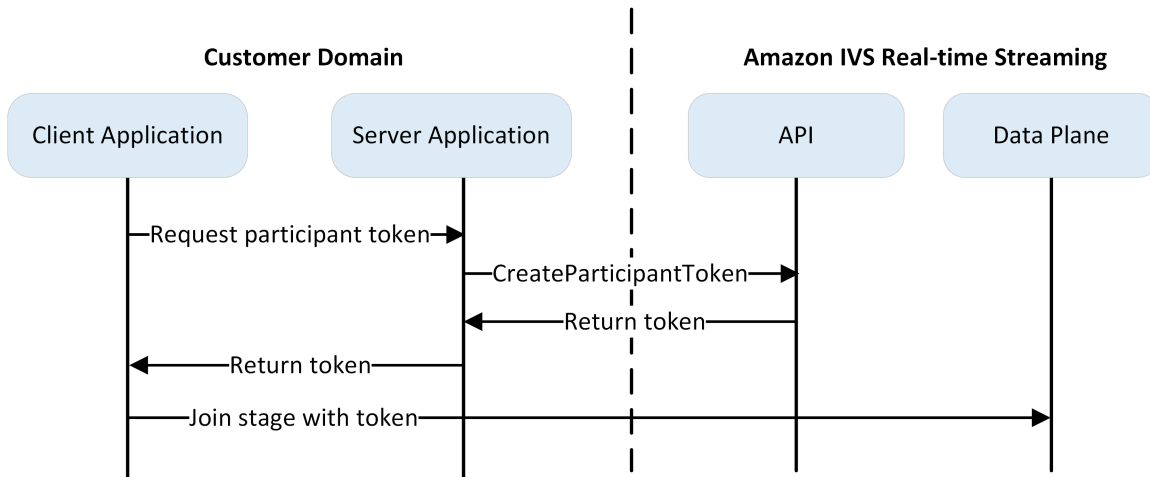
```
ECDSASHA384(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  <private-key>
)
```

指示

1. 使用 ES384 簽署演算法和與提供給 IVS 的公有金鑰相關聯的私有金鑰，來產生權杖的簽章。
2. 組合符記。

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

使用 IVS 即時串流 API 建立權杖



如上所示，用戶端應用程式會向您的伺服器端應用程式索取字符，伺服器端應用程式會使用 AWS SDK 或 Sigv4 簽署的請求呼叫 CreateParticipantToken。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，您可以選擇是否指定屬性和/或功能：

- 您可以指定應用程式提供的屬性，以編碼至權杖並連接至舞台。映射的鍵值可以包含 UTF-8 編碼文字。此欄位的長度上限總計為 1 KB。此欄位會向所有舞台參與者顯示，而此資訊不應用於個人身分識別、機密或敏感資訊。
- 您可以指定權杖可啟用的功能。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。

如需詳細資訊，請參閱 [CreateParticipantToken](#)。

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. 選取建立權杖。建立權杖視窗隨即出現。
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。
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/VSWjvX5X0kU3 --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":
    "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJleHAiOjE2NjE1NDE0MjAsImp0aSI6ImpGcFdtVmFTm9sUyIsInJlYyI6IjE2NjE1NDE0MjAsImp0aSI6ImpGcFdtVmFTm9sUyIsInR5cCI6IkpXVCJ9.TaKj1lW9Qac6c5xBrdAk"
  }
}
```

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。以下是使用 JavaScript 的 AWS SDK 說明。

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：若要使用下面的程式碼範例，您需要安裝 `aws-sdk/client-ivs-realtime` 套件。如需詳細資訊，請參閱[適用於 JavaScript 的 AWS SDK 入門](#)。

```
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);
```

步驟 4：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。如果已[為舞台設定 RTMP 擷取](#)，則可以使用任何可以廣播到 RTMP 端點的編碼器 (例如 OBS 或 ffmpeg)。

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。下列步驟將引導您建立一個稱為 `BasicRealTime` 的應用程式。應用程式的完整程式碼位於 CodePen 和 GitHub：

- Web：<https://codepen.io/amazon-ivs/pen/ZEggrpo>
- Android 版：<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

Web 廣播 SDK 以 JavaScript 程式庫的形式發布，可以在 <https://web-broadcast.live-video.net/1.17.0/amazon-ivs-web-broadcast.js> 上擷取。

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 `IVSBroadcastClient` 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:1.23.0: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，請選取應用程式。
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 CocoaPods 整合廣播 SDK。(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
```

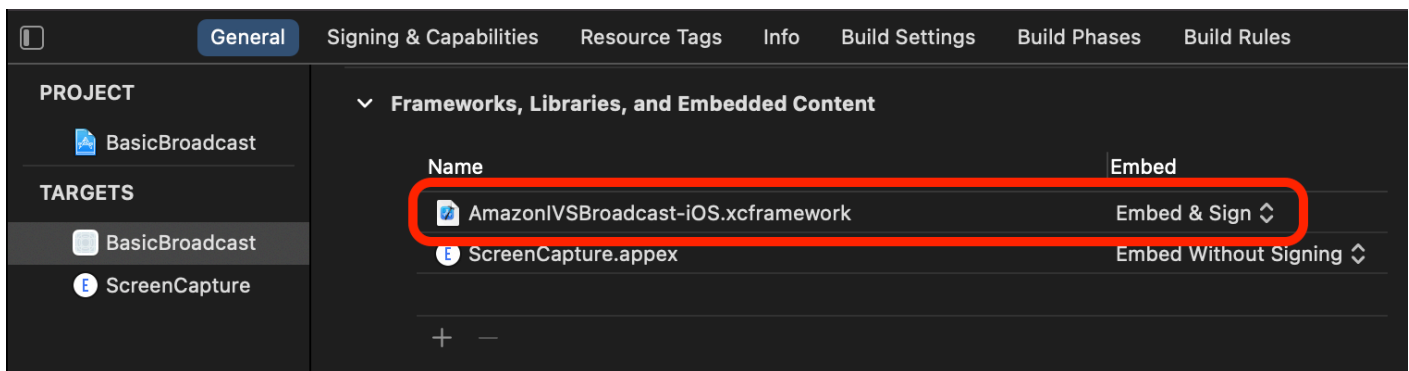


```
# Comment the next line if you don't want to use dynamic frameworks
use_frameworks!

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

替代方法：手動安裝架構

1. 從 <https://broadcast.live-video.net/1.23.0/AmazonIVSBroadcast-Stages.xcframework.zip> 中下載最新版本。
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

步驟 5：發布和訂閱影片

您可以使用下列方式發布到 IVS/訂閱 (即時) IVS：

- 支援 WebRTC 和 RTMPS 的原生 [IVS 廣播 SDK](#)。建議採用此方式，尤其在生產情境下更是如此。請參閱下列有關 [Web](#)、[Android](#) 和 [iOS](#) 的詳細資訊。
- Amazon IVS 主控台：適用於測試串流。請參閱以下。
- 其他串流軟體和硬體編碼器：您可以使用任何支援 RTMP、RTMPS 或 SRT 通訊協定的串流編碼器。如需詳細資訊，請參閱 [串流擷取](#)。

IVS 主控台

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，選取舞台。(如果導覽窗格已收起，選取漢堡圖示即可展開。)
3. 選取您要訂閱或向其進行發布的舞台，以前往其詳細資訊頁面。
4. 若要訂閱：如果舞台有一或多名發布者，您可以按下訂閱索引標籤下的訂閱按鈕來進行訂閱。(索引標籤位於一般組態區段下方)。
5. 若要發布：
 - a. 選取發布索引標籤。
 - b. 系統會提示您授予 IVS 主控台存取攝影機和麥克風的權限；請允許這些權限。
 - c. 在發布索引標籤的底部，使用下拉式方塊選取麥克風和攝影機的輸入裝置。
 - d. 若要開始發布，請選取開始發布。
 - e. 若要檢視已發布的內容，請返回訂閱索引標籤。
 - f. 若要停止發布，請前往發布索引標籤，並將按下底部的停止發布按鈕。

注意：訂閱和發布會耗用資源，而且連線到舞台的時間會產生每小時費用。若要進一步了解，請參閱 IVS 定價頁面上的 [即時串流](#)。

使用 IVS Web 廣播 SDK 發布和訂閱

本節將引導您完成使用 Web 應用程式發布和訂閱階段的相關步驟。

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/1.17.0/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：

IVS Real-Time Streaming

Token

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 JoinStage 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？

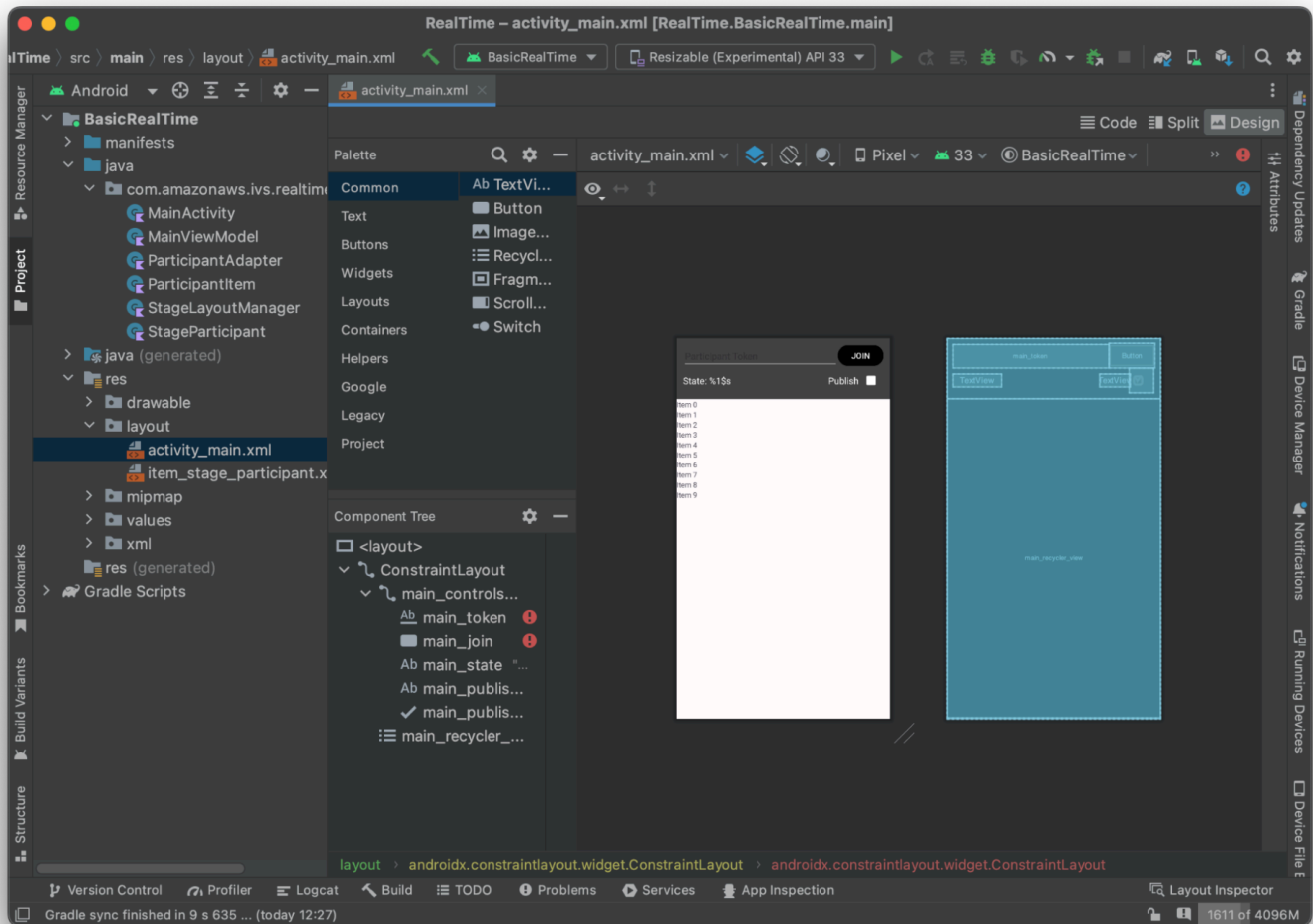
有關 npm、React 等的更詳細範例，請參閱 [IVS 廣播 SDK：網絡指南（即時串流指南）](#)。

使用 IVS Android 廣播 SDK 發布和訂閱

本節將引導您完成使用 Android 應用程式發布和訂閱階段的相關步驟。

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:

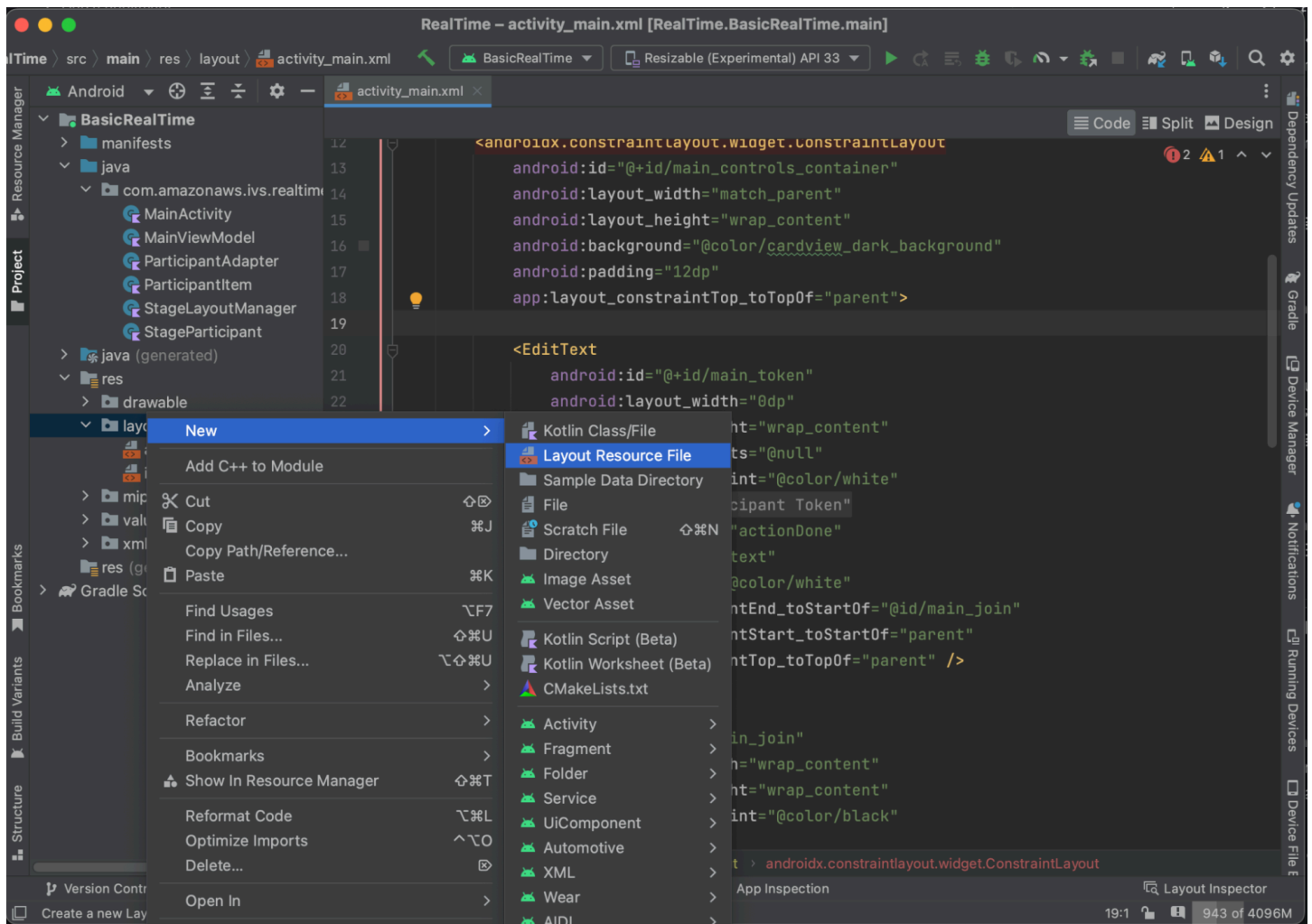
```
import android.widget.Button
import android.widget.CheckBox
import android.widget.EditText
import android.widget.TextView
import androidx.recyclerview.widget.RecyclerView

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

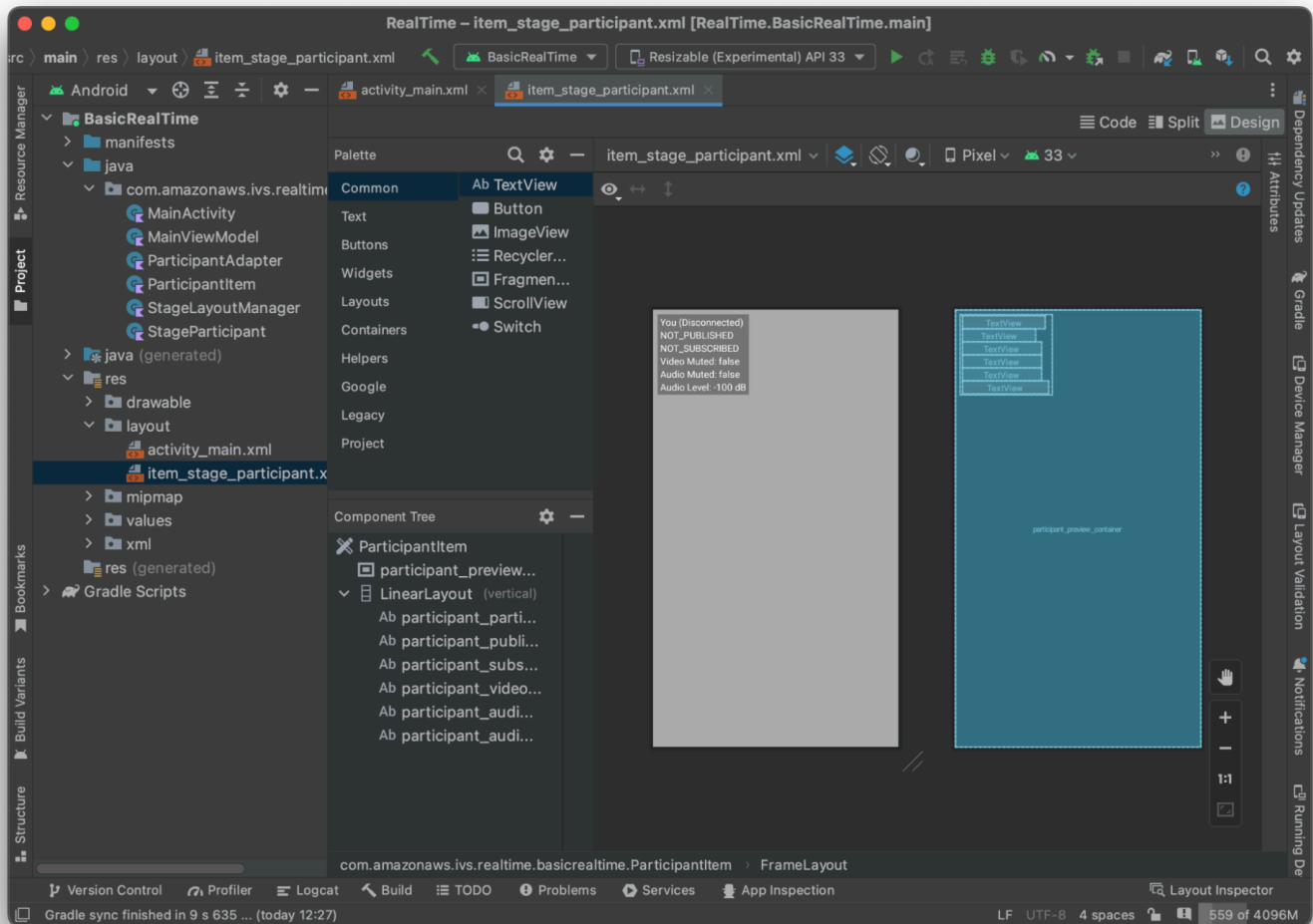
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    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

我們的應用程式會在 `MainViewModel.kt` 中本機追蹤參與者，並且使用 Kotlin 的 [StateFlow](#) 將狀態傳回至 `MainActivity`。

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, Stage.Renderer {

}
```

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 Adapter

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) {}
}
```

實作自訂 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 currentPosition = position
            while (currentPosition - config[row] >= 0) {
                currentPosition -= 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())  
}  
checkboxboxPublish.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
```

使用 IVS iOS 廣播 SDK 發布和訂閱

本節將引導您完成使用 iOS 應用程式發布和訂閱階段的相關步驟。

建立檢視

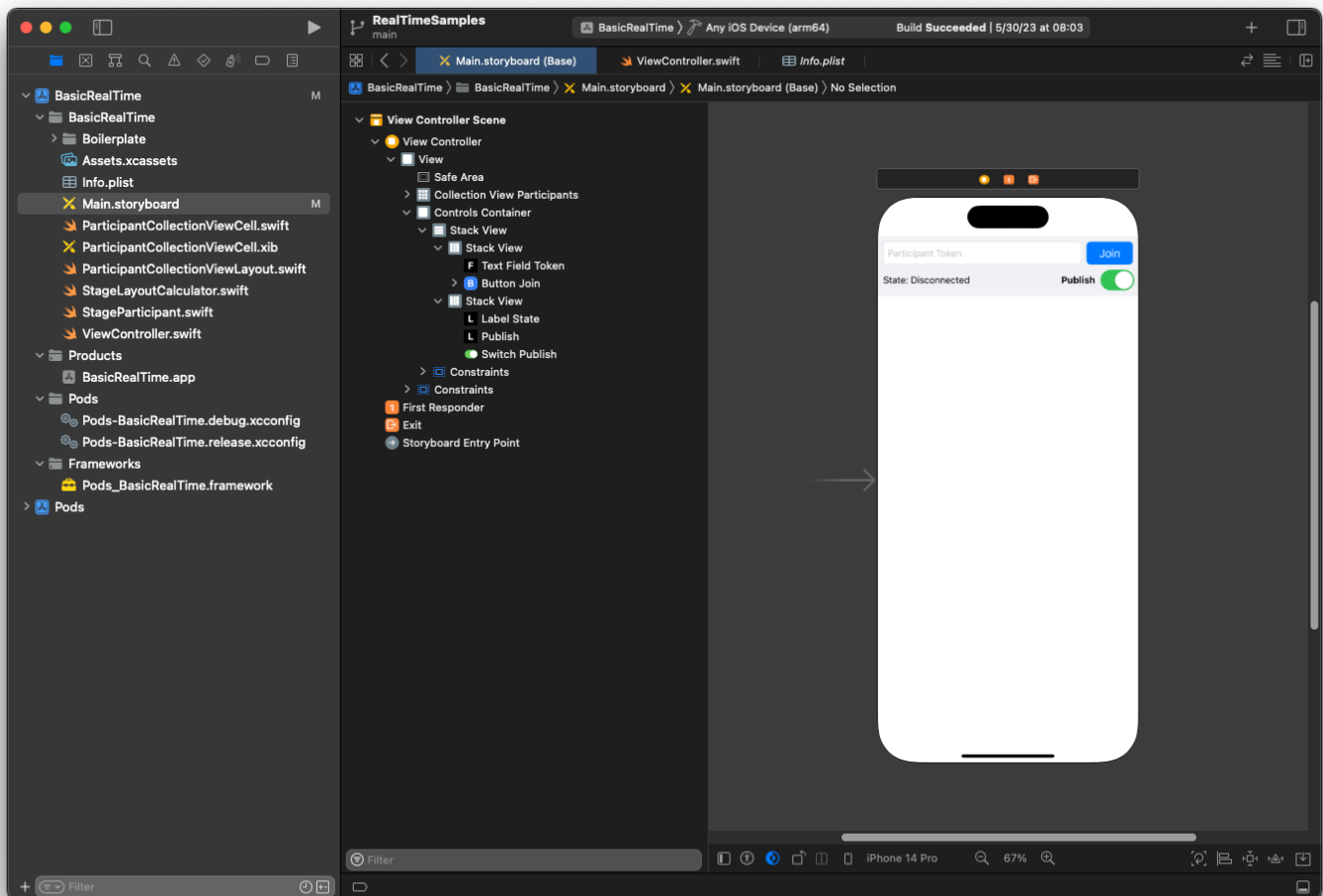
首先，使用自動建立用於匯入 `AmazonIVSBroadcast` 的 `ViewController.swift` 檔案，然後新增一些要連結的 `@IBOutlet`s：

```
import AmazonIVSBroadcast

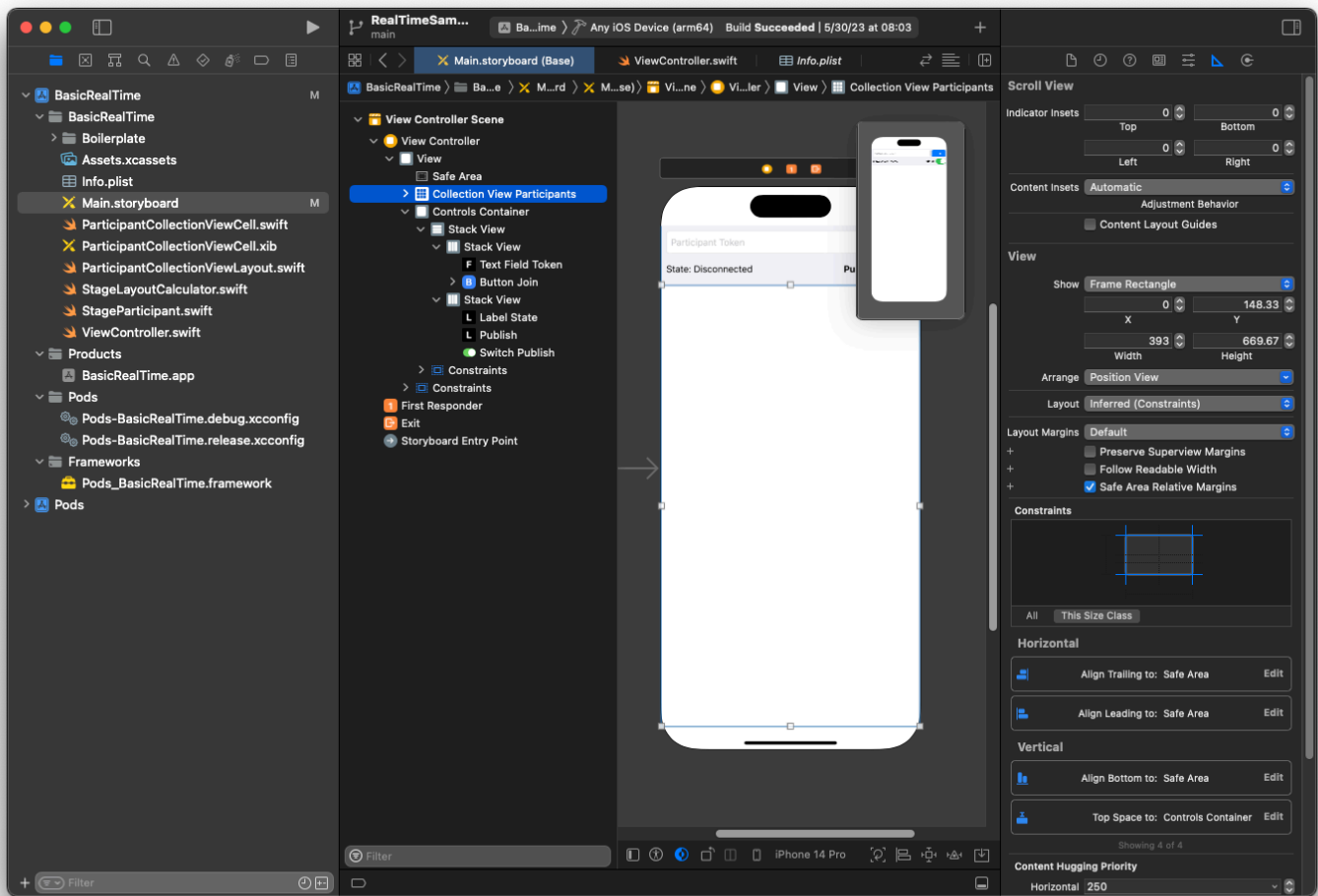
class ViewController: UIViewController {

    @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!
```

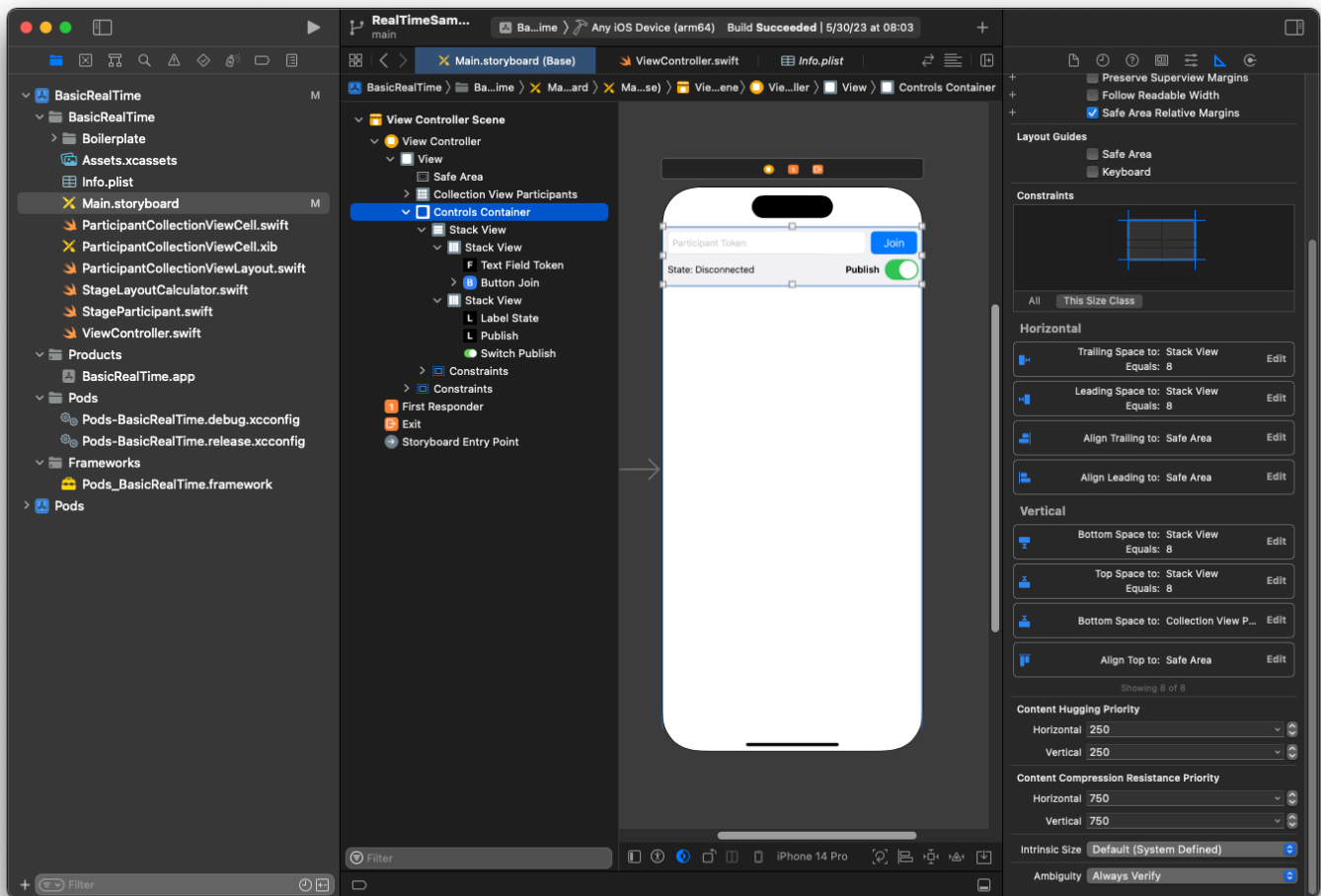
現在，建立這些檢視，並在 Main.storyboard 中將其連結。以下是將使用的檢視結構：



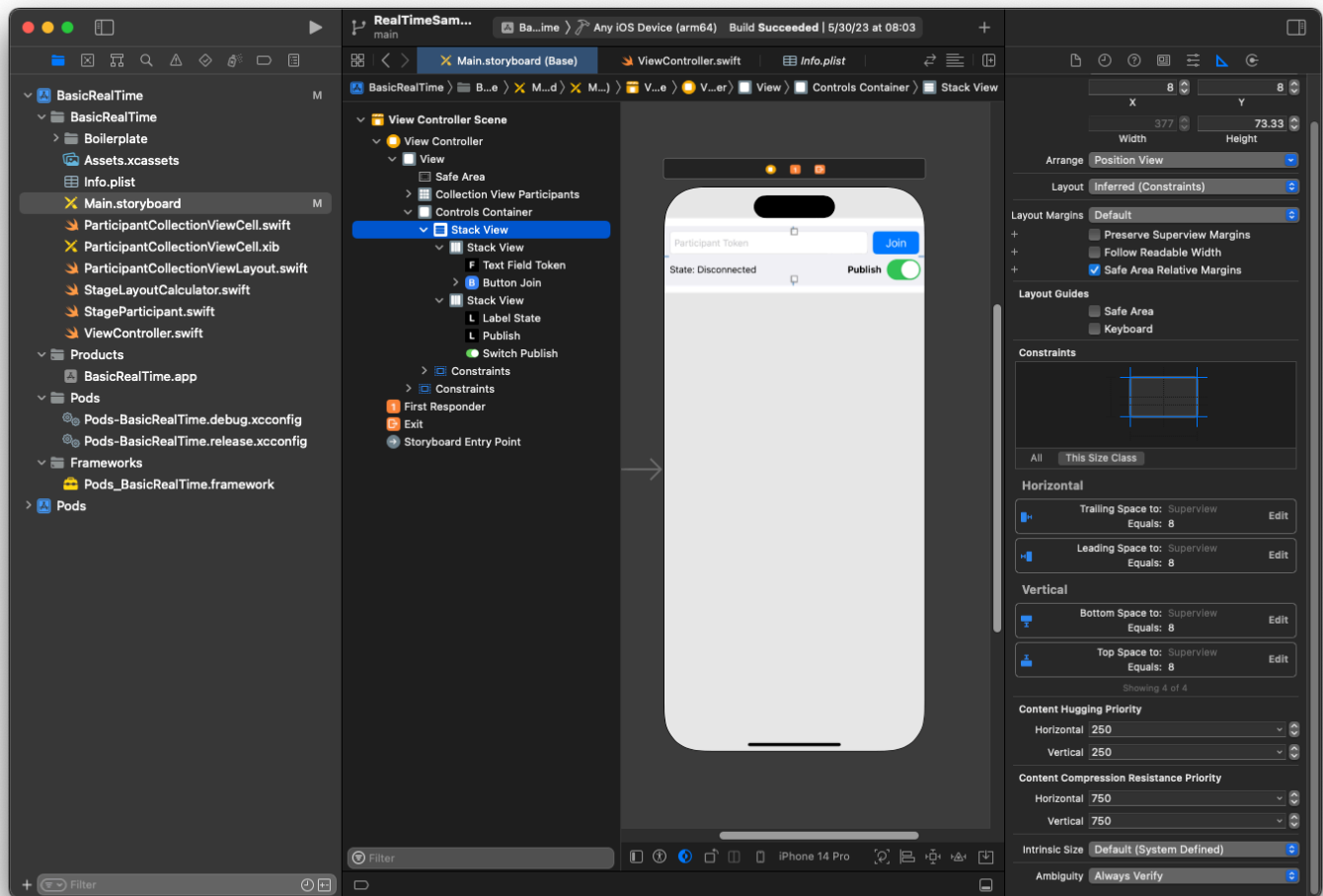
若是 AutoLayout 組態，需要自訂三個檢視。第一個檢視是集合檢視參與者 (UICollectionView)。將前導、尾隨和底部繫結至安全區域。同時將頂部繫結至控制容器。



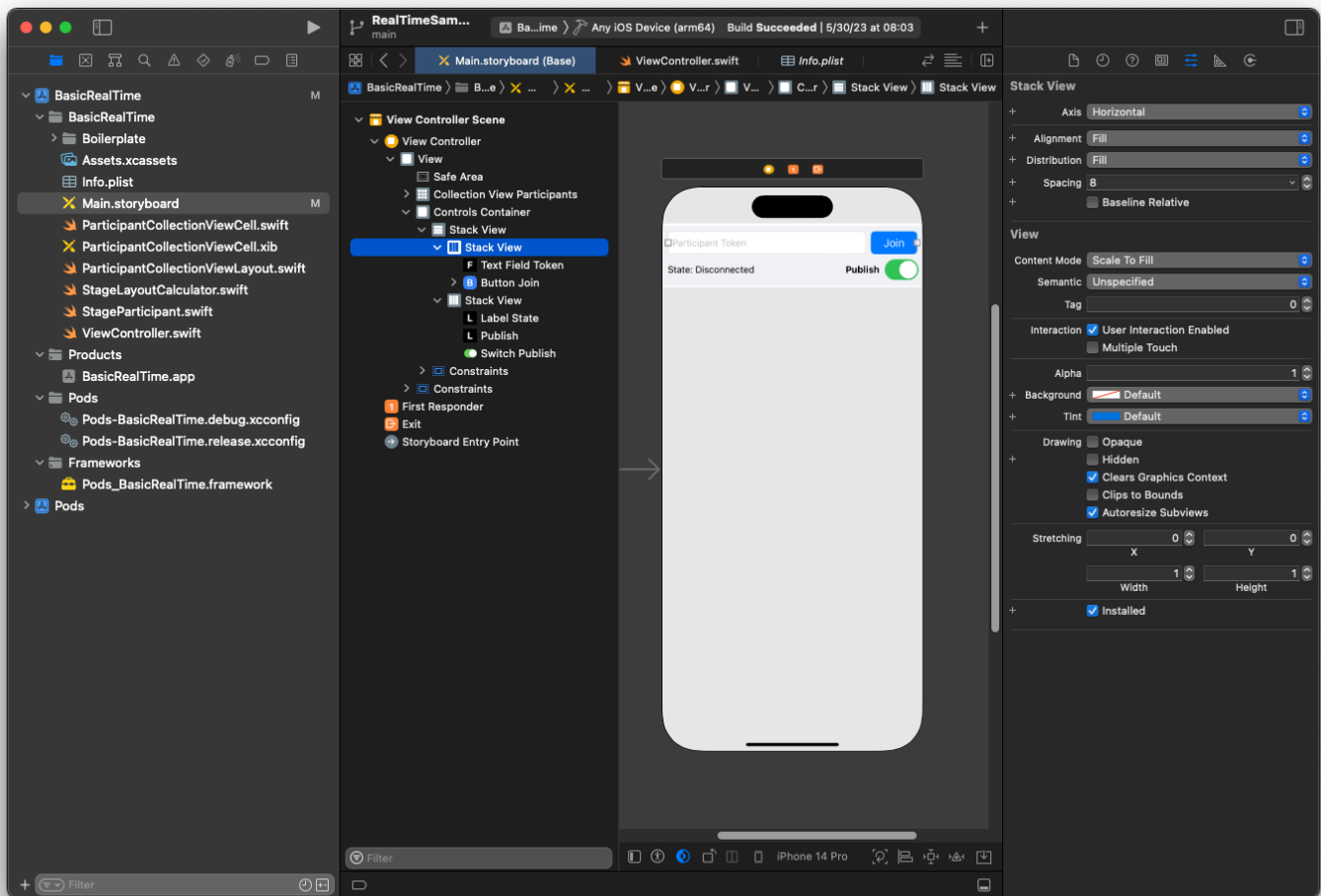
第二個檢視是控制容器。將前導、尾隨和頂部繫結至安全區域。



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。



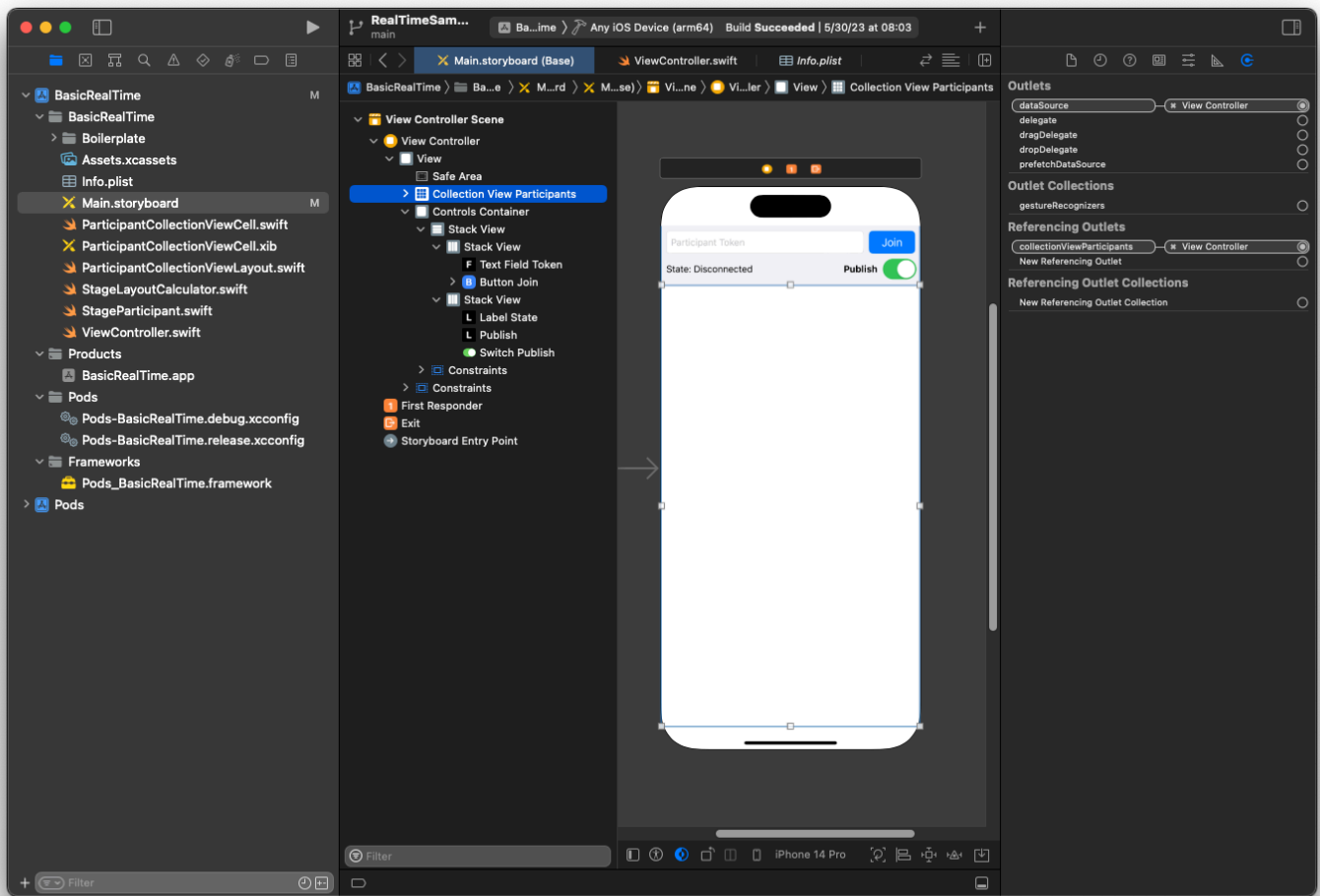
UIStackViews 會處理其餘檢視的配置。針對所有三個 UIStackViews，在對齊和分佈時使用填滿。



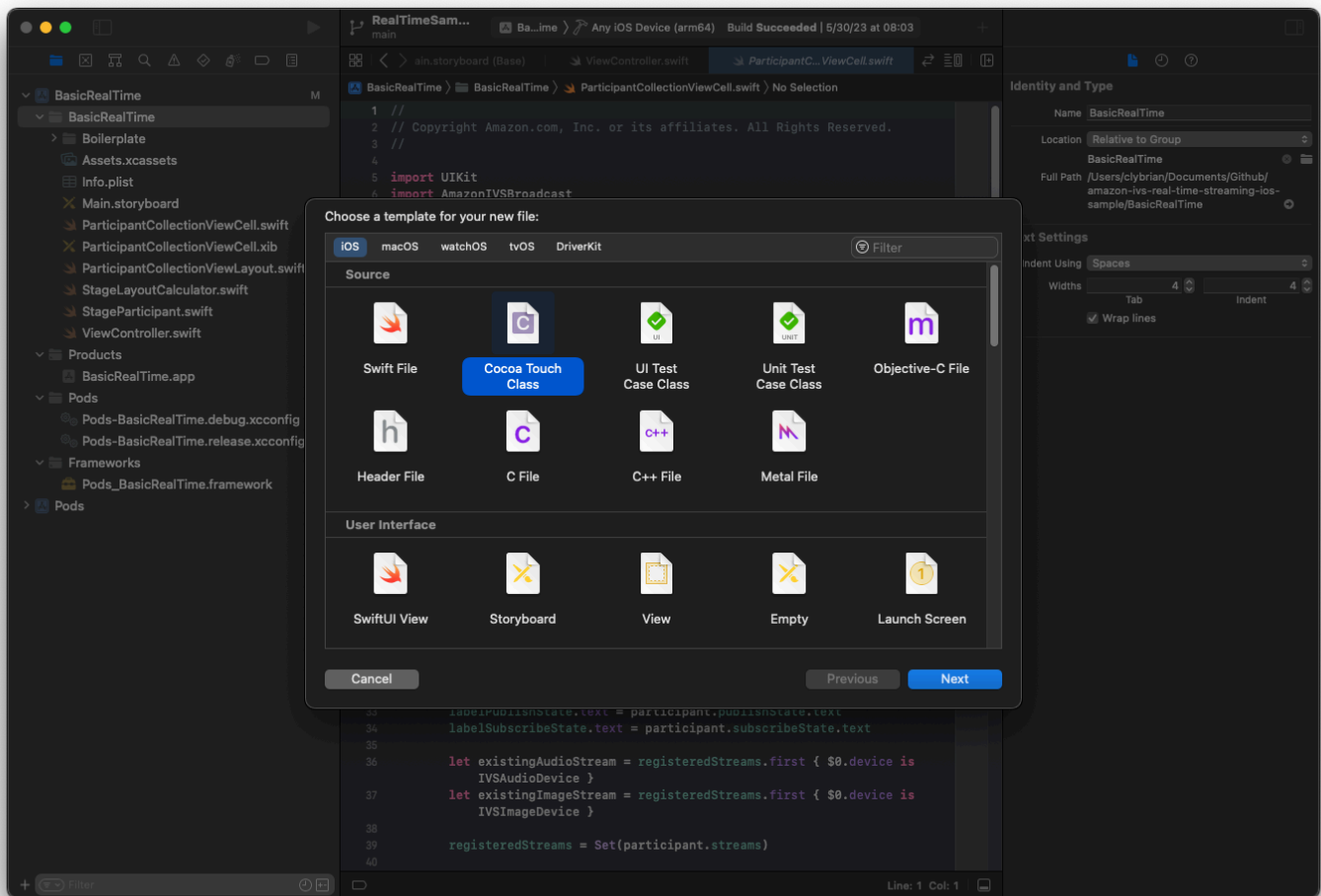
最後，將這些檢視連結至 ViewController。從上方映射下列檢視：

- 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：



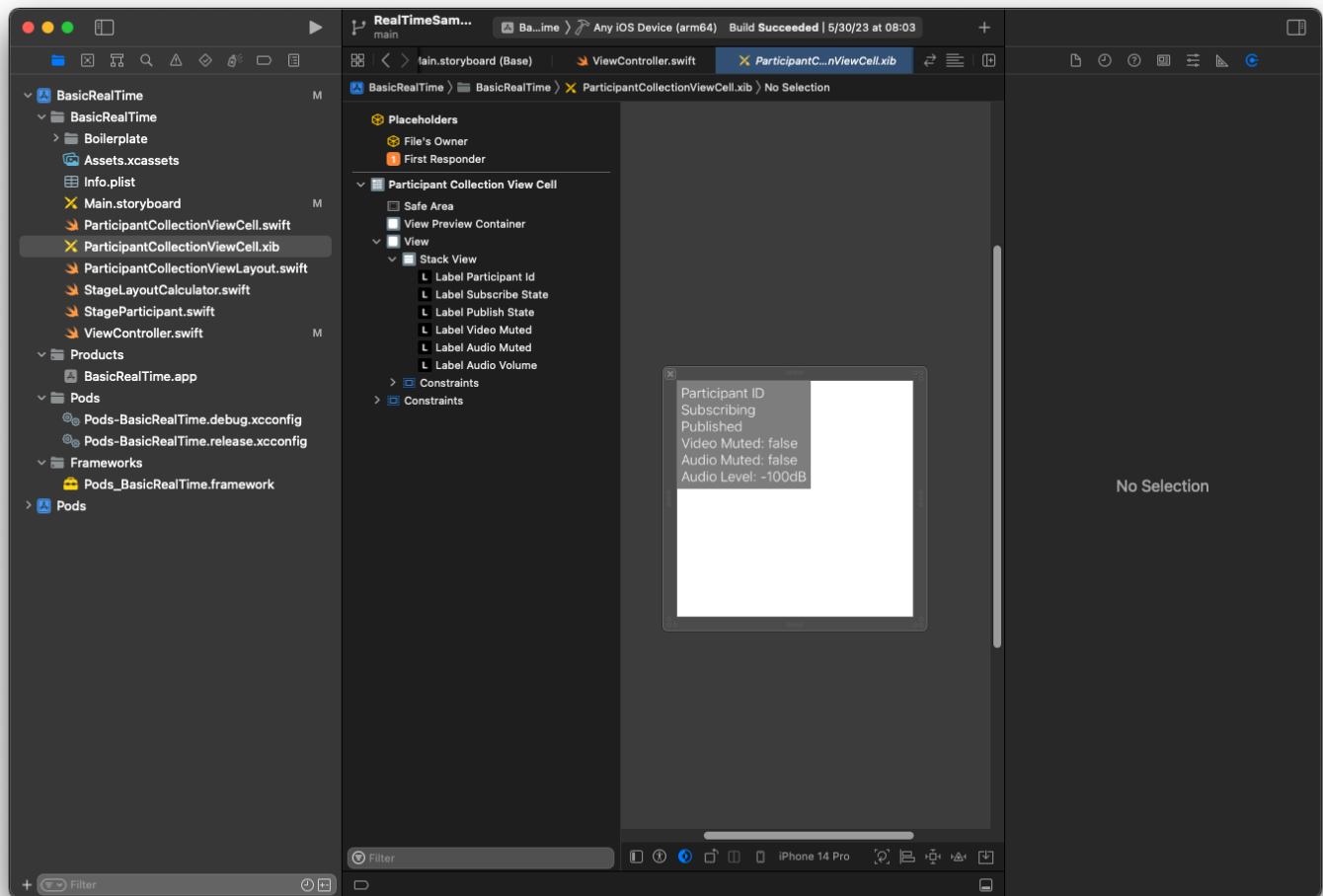
將其命名為 `ParticipantUICollectionViewCell`，然後使其成為 Swift 中 `UICollectionViewCell` 的子類別。再次從 Swift 檔案開始，建立要連結的 `@IBOutlet`：

```
import AmazonIVSBroadcast

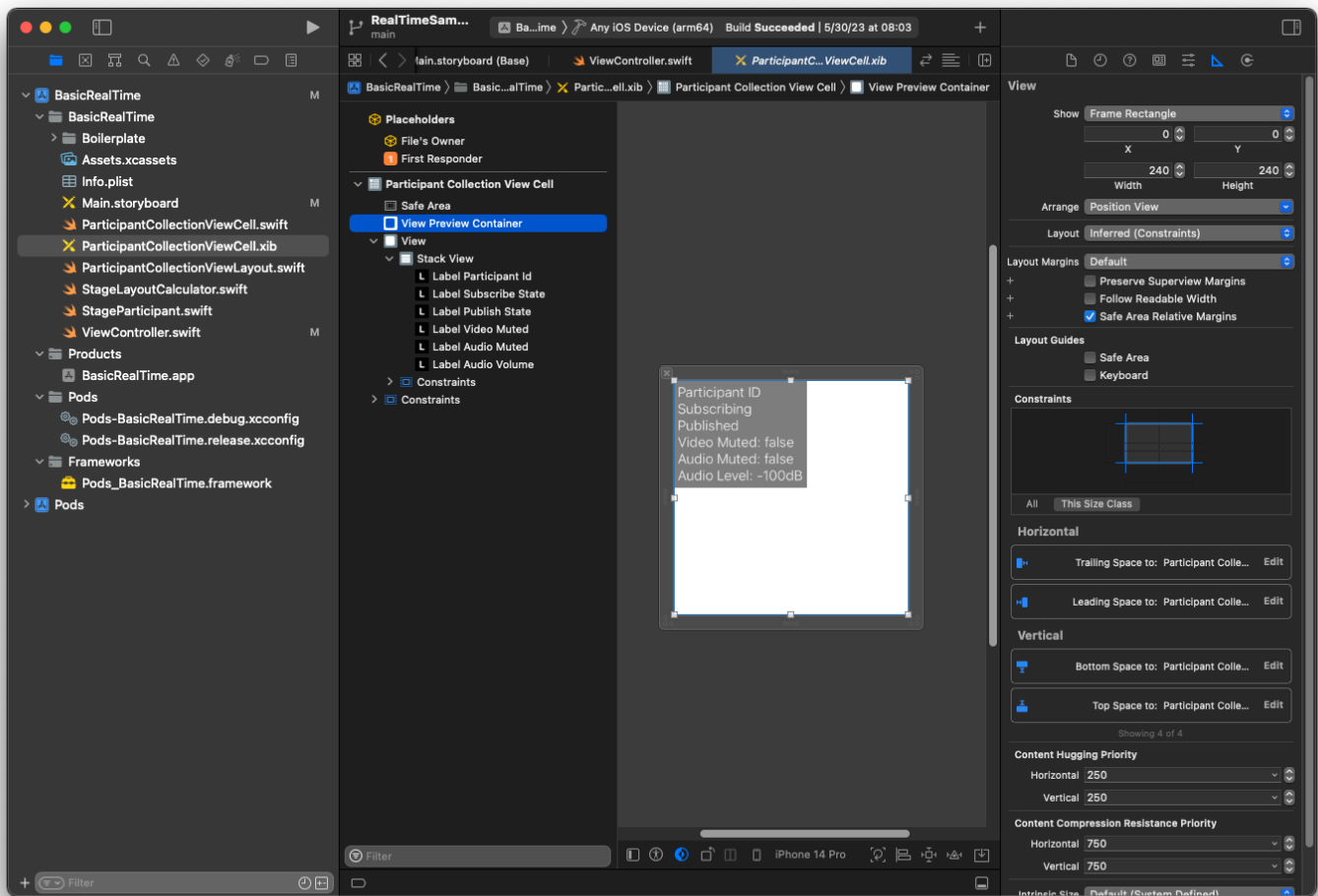
class ParticipantUICollectionViewCell: UICollectionViewCell {

    @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!
```

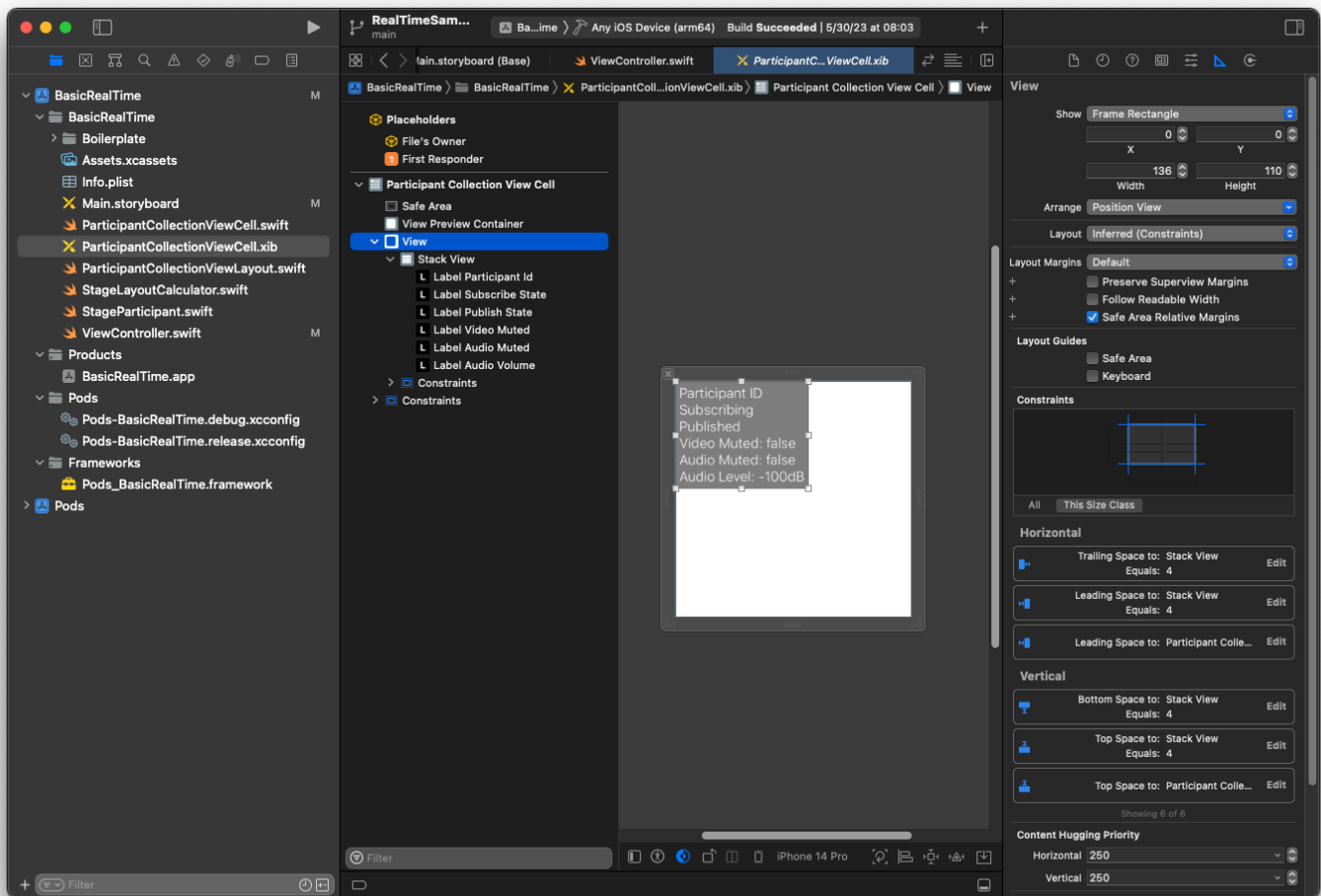
在關聯的 XIB 檔案中，建立此檢視階層：



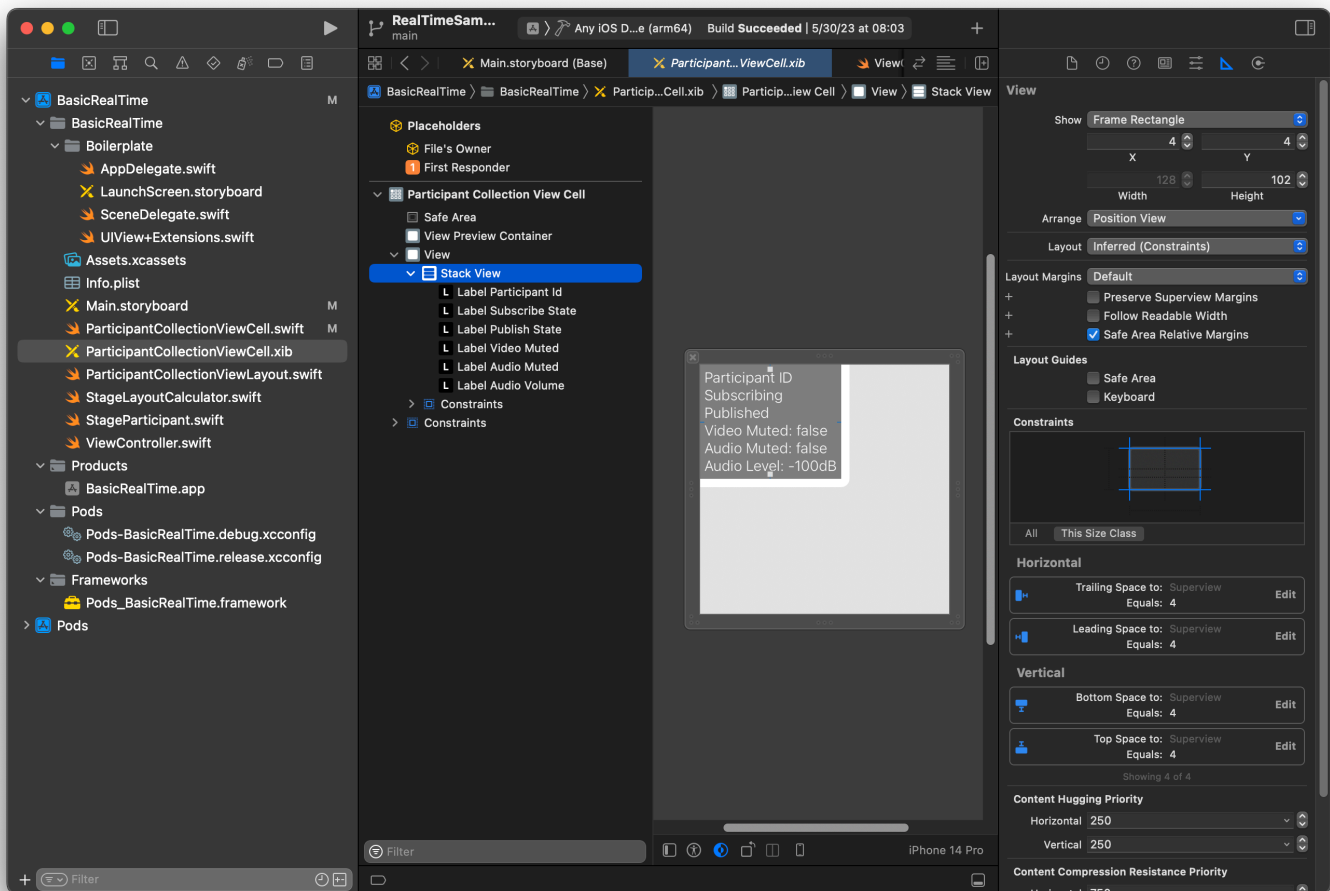
針對 AutoLayout，將再次修改三個檢視。第一個檢視是檢視預覽容器。將尾隨、前導、頂部和底部設定為參與者集合檢視儲存格。



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 viewDidLoad(_ animated: Bool) {
    super.viewDidLoad(animated)
    // Prevent the screen from turning off during a call.
    UIApplication.shared.isIdleTimerDisabled = true
}

override func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(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 reloadData, 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。

IVSStageStrategy

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。

IVSStageRenderer

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 UICollectionViewLayout

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)

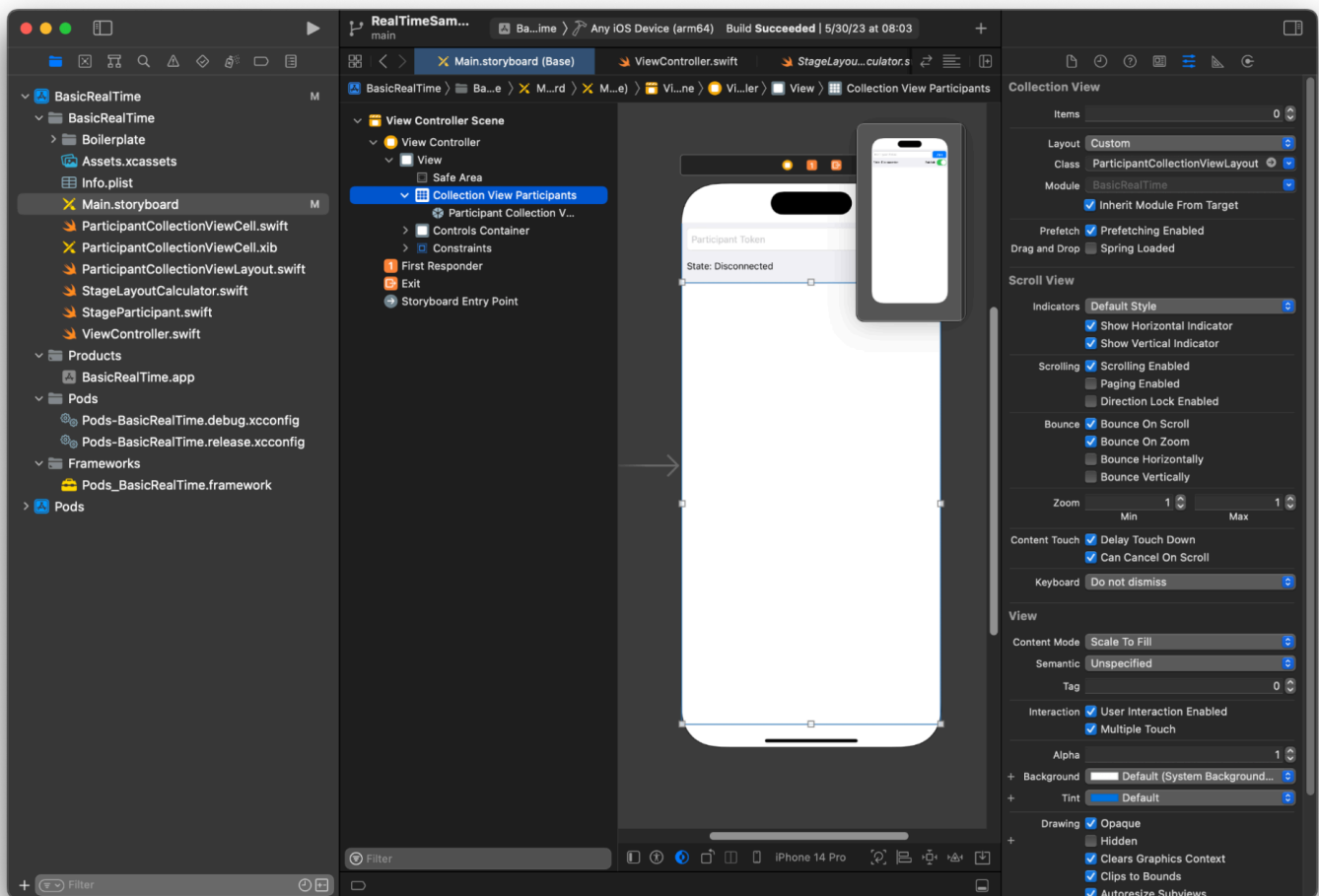
var frames = [CGRect]()
for row in 0 ..< layout.count {
    // layout[row] is the number of columns in a layout
    let itemWidth = (isVertical ? width : height) / CGFloat(layout[row])
    let segmentFrame = CGRect(x: (isVertical ? 0 : lastFrame.maxX) +
halfPadding,
                                y: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

本文件提供有關可用於監控 IVS 即時串流應用程式之選項的詳細資訊。

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。(如果導覽窗格已折疊，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": "AdRezB1021t0"
    },
    {
      "eventTime": "2023-04-04T22:48:41+00:00",
      "name": "SUBSCRIBE_STARTED",
      "participantId": "AdRezB1021t0",
      "remoteParticipantId": "0u5b5n5XLMdC"
    },
    {
      "eventTime": "2023-04-04T22:49:45+00:00",
      "name": "SUBSCRIBE_STOPPED",
      "participantId": "AdRezB1021t0",
      "remoteParticipantId": "0u5b5n5XLMdC"
    },
    {
      "eventTime": "2023-04-04T22:49:45+00:00",
```

```
        "name": "LEFT",
        "participantId": "AdRezB1021t0"
    }
  ]
}
```

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. 透過 <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 命令列介面使用者指南》](#)。

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::- 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	說明
DownloadPacketLoss	Stage	<p>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。</p> <p>單位：百分比</p> <p>有效統計資料：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)</p>
DownloadPacketLoss	Stage, Participant	<p>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。</p> <p>單位：百分比</p>

指標	維度	說明
		有效統計資料：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	<p>每個範例都代表指定訂閱用戶捨棄的影格百分比。</p> <p>單位：百分比</p> <p>有效統計資料：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)</p>
DroppedFrames	Stage, Participant	<p>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。</p> <p>單位：百分比</p> <p>有效統計資料：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)</p>
PublishBitrate	Stage	<p>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)。</p> <p>單位：位元/秒</p> <p>有效統計資料：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)</p>
PublishBitrate	Stage, Participant, Simulcast Layer, MediaType	<p>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。</p> <p>單位：位元/秒</p> <p>有效統計資料：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)</p>

指標	維度	說明
PublishFrameRate	Stage, Participant	<p>從指定發布者接收影片影格的頻率。此指標僅適用於透過 RTMP 進行發布的參與者。</p> <p>單位：個/秒</p> <p>有效統計資料：平均值、最大值、最小值 – 設定間隔內影格速率的平均數、最大數或最小數 (分別)</p>
Publishers	Stage	<p>發布至階段的參與者人數。</p> <p>單位：計數</p> <p>有效統計資料：平均值、最大值、最小值</p>
PublishResolution	Stage, Participant, Simulcast Layer, MediaType	<p>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。</p> <p>單位：計數</p> <p>有效統計資料：平均值、最大值、最小值</p>
SubscribeBitrate	Stage	<p>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。</p> <p>單位：位元/秒</p> <p>有效統計資料：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)</p>
SubscribeBitrate	Stage, Participant, MediaType	<p>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。</p> <p>單位：位元/秒</p> <p>有效統計資料：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)</p>

指標	維度	說明
Subscribers	Stage	<p>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。</p> <p>單位：計數</p> <p>有效統計資料：平均值、最大值、最小值</p>

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 (當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，我們不建議您合成階段並將其廣播到 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 Web 廣播 SDK：1.17.0 ([版本備註](#))

參考文件：如需有關 Amazon IVS Web 廣播 SDK 中可用的最重要方法的資訊，請參閱 <https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference>。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](#)。

開始使用 IVS Web 廣播 SDK | 即時串流

本文件將帶您了解開始使用 IVS 即時串流 Web 廣播 SDK 的相關步驟。

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'
```

伺服器端轉譯支援

無法在伺服器端環境中載入 Web 廣播 SDK 舞台程式庫，因為其在載入時會參考程式庫運作所需的瀏覽器基本概念。若要解決此問題，請依[使用 Next 和 React 的 Web 廣播示範](#)所示，動態載入此程式庫。

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 [Permissions API](#) (許可 API) 和 [MediaDevices.getUserMedia\(\)](#)。

列出可用裝置

若要查看可以擷取的裝置，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');
```

從裝置擷取 MediaStream

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。

對於即時串流，請確保媒體解析度限制為 720p。具體而言，`getUserMedia` 和 `getDisplayMedia` 的寬度和高度限制值相乘後不得超過 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 },
});
```

使用 IVS Web 廣播 SDK 發布和訂閱 | 即時串流

本文件將帶您了解開始使用 IVS 即時串流 Web 廣播 SDK 發布和訂閱階段的相關步驟。

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。例如，假設應用程式在使用 CreateParticipantToken 建立權杖時提供了 role 屬性。此應用程式可能會使用 StageParticipantInfo 的 attributes 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.attributes.role) {
      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。

參與者訂閱組態

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

如果正在訂閱遠端參與者 (請參閱[訂閱參與者](#))，則 SDK 會查詢主機應用程式關於該參與者的自訂訂閱組態。此組態為選用功能，允許主機應用程式控制某些層面的訂閱用戶行為。如需有關可設定內容的詳細資訊，請參閱 SDK 參考文件中的 [SubscribeConfiguration](#)。

以下是實作範例：

```
const strategy = {

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

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

此實作會將所有訂閱參與者的抖動緩衝區最低延遲更新為預設的 MEDIUM。

您也可以透過 `shouldSubscribeToParticipant` 採用更進階的實作方式。指定的 `ParticipantInfo` 可用來專門更新特定參與者的訂閱組態。

我們建議您使用預設行為。只有在您想要變更特定行為時，才指定自訂組態。

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

使用與上述 [從裝置擷取 MediaStream](#) 中相同的步驟來擷取麥克風和攝影機等本機裝置。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。發布螢幕共用需要為此舞台建立額外的權杖，具體而言，是為發布螢幕共用的媒體而建立權杖。使用 `getDisplayMedia` 並將解析度限制為最高 720p。之後的步驟就與將攝影機發布到此舞台類似。

```
// 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](#) 以取得有關音訊或視訊是否靜音的狀態資訊：

```
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.ERRORRED:
      // SDK encountered an error and lost its connection to the stage. Wait for
      CONNECTED.
      break;
  })
```

一般而言，您可以忽略成功加入舞台後遇到的錯誤狀態，因為 SDK 會嘗試在內部復原。如果 SDK 報告 `ERRORRED` 狀態，且舞台長時間 (例如 30 秒或更久) 保持在 `CONNECTING` 狀態，則表示網路的連線可能已中斷。

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](#)。

IVS Web 廣播 SDK 中的已知問題和解決方法 | 即時串流

本文件列出您在使用 Amazon IVS 即時串流功能 Web 廣播 SDK 時可能遇到的已知問題，並建議潛在的解決方法。

- 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。

- 您可能難以維持在 `StageParticipantPublishState.PUBLISHED` 狀態，而且在接聽 `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：調用 `getUserMedia` 或 `getDisplayMedia` 時，將影片解析度限制為 720p。具體而言，`getUserMedia` 和 `getDisplayMedia` 的寬度和高度限制值相乘後不得超過 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。啟用許可之後，使用者必須重新啟動 Firefox，Firefox 才能正常運作；否則，如果認為許可受到封鎖，瀏覽器會擲回 [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

- 行動裝置上不支援 [getDisplayMedia](#) 螢幕共享。

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。

- 背景處理瀏覽器會導致發布串流時螢幕變成黑色，並產生純音訊內容。

解決方法：無。這是為安全起見。

IVS Web 廣播 SDK 中的錯誤處理 | 即時串流

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。SDK 會將錯誤報告給 `StageEvents.ERROR` 事件的接聽程式：

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

舞台錯誤

當 SDK 遇到無法復原的問題，且通常需要應用程式介入和/或網路重新連線才能復原時，則會報告 `StageError`。

每個報告的 `StageError` 都包含代碼 (或 `StageErrorCode`)、訊息 (字串) 和類別 (`StageErrorCategory`)。每個錯誤都有基礎操作類別。

錯誤的操作類別取決於錯誤與下列哪項相關：舞台 (`JOIN_ERROR`) 連線、將媒體傳送至舞台 (`PUBLISH_ERROR`) 或從舞台 (`SUBSCRIBE_ERROR`) 接收傳入媒體串流。

`StageError` 的代碼屬性會報告特定問題：

名稱	代碼	建議的動作
TOKEN_MALFORMED	1	建立有效權杖，然後重試將舞台執行個體化。
TOKEN_EXPIRED	2	建立未過期的權杖，然後重試將舞台執行個體化。
TIMEOUT	3	此操作逾時。如果舞台存在且權杖是有效的，則此失敗可能是網路問題。若是如此，請等待裝置的連線復原。
失敗	4	嘗試操作時遇到嚴重狀況。檢查錯誤詳細資訊。 如果舞台存在且權杖是有效的，則此失敗可能是網路問題。若是如此，請等待裝置的連線復原。
CANCELED	5	檢查應用程式程式碼，並確保沒有重複的 <code>join</code> 、 <code>refreshStrategy</code> 或 <code>replaceStrategy</code> 調用，因為這可能會導致在操作完成之前啟動和取消重複的操作。
STAGE_AT_CAPACITY	6	重新整理策略，在舞台人數不再滿時，再次嘗試操作。
CODEC_MISMATCH	7	此舞台不支援編解碼器。檢查瀏覽器和平台的編解碼器支援。若是 IVS 即時串流，瀏覽器必須支援 H.264 影片編解碼器和 Opus 音訊編解碼器。
TOKEN_NOT_ALLOWED	8	此權杖沒有該操作的許可。使用正確的許可重新建立權杖，然後再試一次。

處理 StageError 範例

使用 StageError 代碼，來判斷此錯誤是否因權杖過期造成：

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

已加入時出現網路錯誤

如果裝置的網路連線中斷，SDK 可能會失去與舞台伺服器的連線。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。

錯誤狀態

建議您使用這些狀態來記錄應用程式，並向使用者顯示訊息，向他們提醒特定參與者與此舞台的連線問題。

發佈

SDK 會在發布失敗時報告 `ERRORED`。

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

訂閱

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

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` 套件會實作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

最新版 Android 廣播 SDK：1.23.0 ([版本備註](#))

參考文件：如需有關 Amazon IVS Android 廣播 SDK 中最重要方法的資訊，請參閱參考文件，網址為 <https://aws.github.io/amazon-ivs-broadcast-docs/1.23.0/android/>。

範本程式碼：請參閱 GitHub 上的 Android 範本儲存庫：<https://github.com/aws-samples/amazon-ivs-broadcast-android-sample>。

平台需求：Android 9.0 及更高版本。

開始使用 IVS Android 廣播 SDK | 即時串流

本文件將帶您了解開始使用 IVS 即時串流 Android 廣播 SDK 的相關步驟。

安裝程式庫

有數種方法可將 Amazon IVS Android 廣播程式庫新增至 Android 開發環境：直接使用 Gradle、使用 Gradle 版本目錄，或手動安裝 SDK。

直接使用 Gradle：如下所示，將程式庫新增至模組的 `build.gradle` 檔案 (適用於最新版 IVS 廣播 SDK)：

```
repositories {
    mavenCentral()
}

dependencies {
    implementation 'com.amazonaws:ivs-broadcast:1.23.0:stages@aar'
}
```

使用 Gradle 版本目錄：首先將此目錄加入模組的 `build.gradle` 檔案：

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

然後在 `libs.version.toml` 檔案中加入下列項目 (適用於最新版 IVS 廣播 SDK)：

```
[versions]
ivs="1.23.0"

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

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`。

也允許 SDK 控制擴音電話：無論您選擇哪種安裝方法，也請將下列許可新增至資訊清單，以允許 SDK 啟用和停用擴音電話：

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

使用包含偵錯符號的 SDK

我們也發布了包含偵錯符號的廣播 SDK Android 版本。如果在 IVS 廣播 SDK 中遇到當機 (即 `libbroadcastcore.so`)，您可以使用此版本來改善 Firebase Crashlytics 中偵錯報告 (堆疊追蹤) 的品質。向 IVS SDK 團隊報告這些當機案例，可以提升堆疊追蹤的品質，協助更輕鬆地修正問題。

若要使用此版本的 SDK，請將下列項目放入 Gradle 建置檔案：

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

使用上一行，而非此行：

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

將符號上傳至 Firebase Crashlytics

確保已針對 Firebase Crashlytics 設定 Gradle 建置檔案。遵循此處的 Google 指示：

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

請務必加入 `com.google.firebase:firebase-crashlytics-ndk` 作為相依性。

建置要發布的應用程式時，Firebase Crashlytics 外掛程式應自動上傳符號。若要手動上傳符號，請執行下列任一動作：

```
gradle uploadCrashlyticsSymbolFileRelease
```

```
./gradlew uploadCrashlyticsSymbolFileRelease
```

(自動和手動上傳符號兩次，也不會造成損害。)

防止發布 .apk 變大

在封裝發布 .apk 檔案之前，Android Gradle 外掛程式會自動嘗試從共用程式庫 (包括 IVS 廣播 SDK 的 `libbroadcastcore.so` 程式庫) 去除偵錯資訊。不過，有時會沒有這樣做。因此，.apk 檔案可能會變大，而且您可能會收到來自 Android Gradle 外掛程式的警告訊息，表示無法去除偵錯符號，並且會依原樣封裝 .so 檔案。如果發生這種情況，請執行下列操作：

- 安裝 Android NDK。任何最新的版本都可以。
- 將 `ndkVersion <your_installed_ndk_version_number>` 新增至應用程式的 `build.gradle` 檔案。即使應用程式本身不包含原生程式碼，也請執行此操作。

如需詳細資訊，請參閱[問題報告](#)。

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();
    }
}
}

```

使用 IVS Android 廣播 SDK 發布和訂閱 | 即時串流

本文件將帶您了解開始使用 IVS 即時串流 Android 廣播 SDK 發布和訂閱階段的相關步驟。

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。

參與者訂閱組態

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

如果正在訂閱遠端參與者 (請參閱[訂閱參與者](#))，則 SDK 會查詢主機應用程式關於該參與者的自訂訂閱組態。此組態為選用功能，允許主機應用程式控制某些層面的訂閱用戶行為。如需有關可設定內容的詳細資訊，請參閱 SDK 參考文件中的 [SubscribeConfiguration](#)。

以下是實作範例：

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

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

    return config;
}
```

此實作會將所有訂閱參與者的抖動緩衝區最低延遲更新為預設的 `MEDIUM`。

您也可以透過 `shouldSubscribeToParticipant` 採用更進階的實作方式。指定的 `ParticipantInfo` 可用來專門更新特定參與者的訂閱組態。

我們建議您使用預設行為。只有在您想要變更特定行為時，才指定自訂組態。

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 `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) {
        frontCamera = 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;
    subscribeType = 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 由於未知網路錯誤而中斷」
- 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);
```

IVS Android 廣播 SDK 中的已知問題和解決方法 | 即時串流

本文件列出您在使用 Amazon IVS 即時串流功能 Android 廣播 SDK 時可能遇到的已知問題，並建議潛在的解決方法。

- 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。

- 具有 MediaTek Helio 晶片組的裝置可能會無法正確轉譯遠端參與者的影片。

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 Android 影片編碼器無法設定為小於 176x176 的影片大小。設定較小的大小會導致錯誤且無法進行串流。

因應措施：請勿將影片大小設定為小於 176x176。

IVS Android 廣播 SDK 中的錯誤處理 | 即時串流

本節概述錯誤情況、IVS 即時串流 Android 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。

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)`
- `MultihostSessionNolceCandidates (1022)`
- `MultihostSessionStageAtCapacity (1024)`
- `SignallingSessionCannotRead (1201)`
- `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

最新版 iOS 廣播 SDK：1.23.0 ([版本備註](#))

參考文件：如需有關 Amazon IVS iOS 廣播 SDK 中最重要方法的資訊，請參閱參考文件，網址為 <https://aws.github.io/amazon-ivs-broadcast-docs/1.23.0/ios/>。

範本程式碼：請參閱 GitHub 上的 iOS 範本儲存庫：<https://github.com/aws-samples/amazon-ivs-broadcast-ios-sample>。

平台需求：iOS 14 或更高版本。

開始使用 IVS iOS 廣播 SDK | 即時串流

本文件將帶您了解開始使用 IVS 即時串流 iOS 廣播 SDK 的相關步驟。

安裝程式庫

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

建議：整合廣播開發套件 (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。

透過 CocoaPods 用名稱 AmazonIVSBroadcast 發行版本。將此相依性新增到您的 Podfile：

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

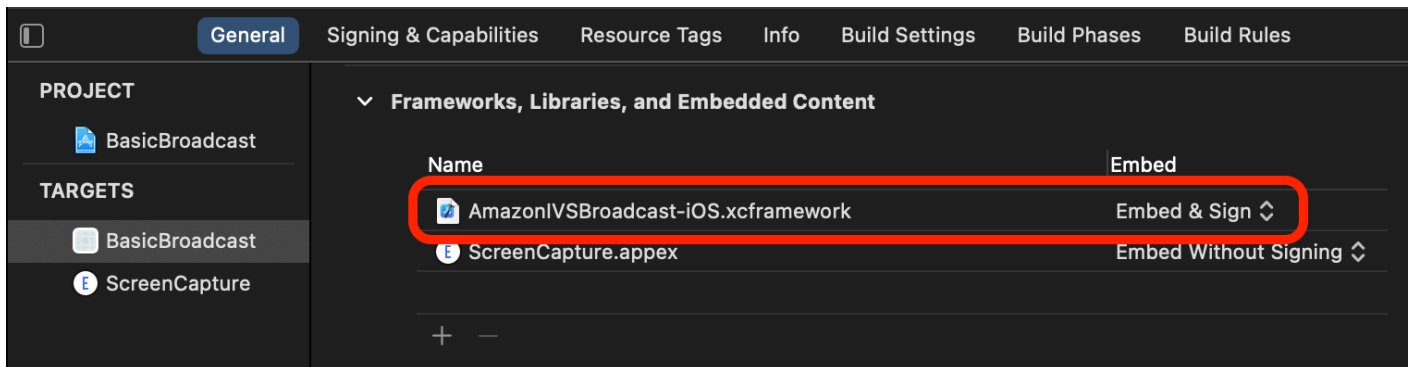
執行 `pod install`，將可在您的 `.xcworkspace` 中使用開發套件。

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。若透過 CocoaPods 將階段子規格新增至您的專案，請務必移除 Podfile 中含有 AmazonIVSBroadcast 的其他行。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. 從 <https://broadcast.live-video.net/1.23.0/AmazonIVSBroadcast-Stages.xcframework.zip> 中下載最新版本。
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。



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
}
```

使用 IVS iOS 廣播 SDK 發布和訂閱 | 即時串流

本文件將帶您了解開始使用 IVS 即時串流 iOS 廣播 SDK 發布和訂閱階段的相關步驟。

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。

參與者訂閱組態

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

如果正在訂閱遠端參與者 (請參閱[訂閱參與者](#))，則 SDK 會查詢主機應用程式關於該參與者的自訂訂閱組態。此組態為選用功能，允許主機應用程式控制某些層面的訂閱用戶行為。如需有關可設定內容的詳細資訊，請參閱 SDK 參考文件中的 [SubscribeConfiguration](#)。

以下是實作範例：

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

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

    return config
  }
```

此實作會將所有訂閱參與者的抖動緩衝區最低延遲更新為預設的 MEDIUM。

您也可以透過 `shouldSubscribeToParticipant` 採用更進階的實作方式。指定的 `ParticipantInfo` 可用來專門更新特定參與者的訂閱組態。

我們建議您使用預設行為。只有在您想要變更特定行為時，才指定自訂組態。

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：

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

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

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

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

廣播 SDK 管理的攝影機會優化其解析度和影格速率 (每秒影格數或 FPS)，以將溫升和能耗降至最低。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。

IVS iOS 廣播 SDK 中的已知問題和解決方法 | 即時串流

本文件列出您在使用 Amazon IVS 即時串流功能 iOS 廣播 SDK 時可能遇到的已知問題，並建議潛在的解決方法。

- 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。

IVS iOS 廣播 SDK 中的錯誤處理 | 即時串流

本節概述錯誤情況、IVS 即時串流 iOS 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。

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。

SDK 會擲出一個 Swift 例外狀況，包含 `error code = 1000` 及 `IVSBroadcastErrorIsFatalKey = YES`。

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 會擲出一個 Swift 例外狀況，包含 `error code = 1001` 及 `IVSBroadcastErrorIsFatalKey = YES`。

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 `stage(didChange connectionState, withError error)` 包含 `error code = 1026` 及 `IVSBroadcastErrorIsFatalKey = YES`。

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 `stage(didChange connectionState, withError error)` 包含 `error code = 1300` 及 `IVSBroadcastErrorIsFatalKey = YES`。

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 `stage(didChange connectionState, withError error)` 包含 `error code = 1300` 及 `IVSBroadcastErrorIsFatalKey = YES`。

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)`
- `MultihostSessionNoIcCandidates (1022)`
- `MultihostSessionStageAtCapacity (1024)`
- `SignallingSessionCannotRead (1201)`
- `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 `IVSErrorDelegate:didEmitError` 時出現相關的錯誤代碼，且 `IVSBroadcastErrorIsFatalKey == NO`。

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` 包含 `error code = 1400` 及 `IVSBroadcastErrorIsFatalKey = NO`。

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`。

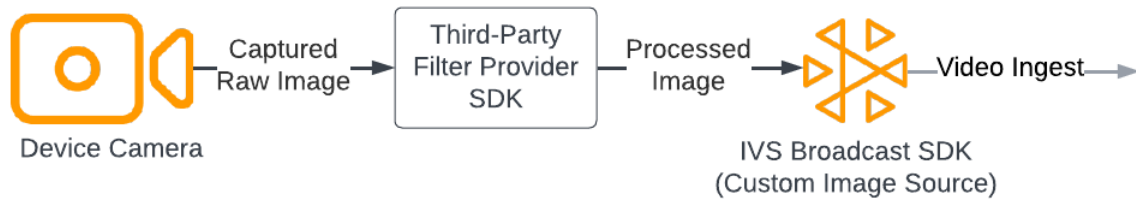
IVS 廣播 SDK：第三方攝影機濾鏡 | 即時串流

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，並發布到階段。
- 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。

搭配 IVS 廣播 SDK 使用 BytePlus

本文說明如何搭配 IVS 廣播 SDK 使用 BytePlus Effects SDK。

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 會調用 `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);
```

搭配 IVS 廣播 SDK 使用 DeepAR

本文說明如何搭配 IVS 廣播 SDK 使用 DeepAR SDK。

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](#)。

搭配 IVS 廣播 SDK 使用 Snap

本文說明如何搭配 IVS 廣播 SDK 使用 Snap 的攝影機套件 SDK。

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。

- 擷取並套用鏡頭。
- 將攝影機套件工作階段的輸出轉譯至畫布。
- 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/LowLatencyUserGuide/multiple-hosts.html">Use the AWS
CLI</a></b> to create a <b>Stage</b> and a corresponding <b>ParticipantToken</b>.
Multiple participants can load this page and put in their own tokens. You can <b><a
href="https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-guides/stages#glossary"
target="_blank">read more about stages in our public docs.</a></b></p>
  </header>
  <hr />

  <!-- Setup Controls -->

  <!-- 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階段的邏輯。)

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階段的邏輯。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階段的事件接聽程式。在用於加入階段的事件接聽程式中，我們從畫布傳遞攝影機套件工作階段和 `MediaStream`，以便將其發布到階段。

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階段。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階段：

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，您可以如下所示，從命令列使用 Python 的 HTTP 伺服器：

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

如果您使用的是 Python 的 HTTP 伺服器，請開啟瀏覽器並前往 `http://localhost:8000`。然後，您應該會在畫面上看到您先前指定且已套用至攝影機供稿的 Snap 鏡頭 AR 效果。

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 [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)

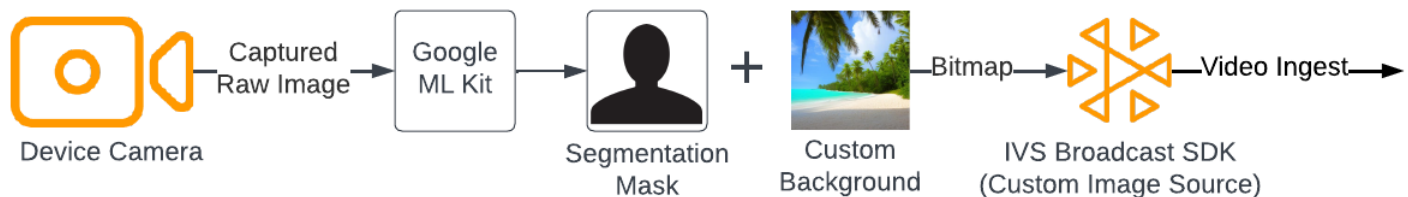
@Override
fun stageStreamsToPublishForParticipant(stage: Stage, participantInfo:
ParticipantInfo): List<LocalStageStream> = publishStreams

```

搭配 IVS 廣播 SDK 使用背景替換

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` 指定為建置指令碼：

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階段的程式碼：

```
<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調用回呼函數。為加入和離開階段按鈕新增事件接聽程式。

請注意，加入階段時，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階段的 `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調用。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，它會調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_segementer/selfie_segementer/float16/latest/selfie_segementer.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蓋到自訂背景上。
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) 就能發布至階段。如需如何設定階段的說明，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)
```

此函數會反復執行遮罩，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階段。

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。

音量鍵類別是指透過裝置上的實體音量鍵使用或變更的音量類型 (媒體音量或通話音量)。請注意，這會影響切換音訊模式時的音量。例如，假設裝置設定為使用視訊聊天預設值時音量為最大值。切換到「僅訂閱」預設值，會導致與作業系統不同的音量，這可能會導致裝置上的音量大幅改變。

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。

iOS 的已知問題：使用此預設值且不連接麥克風，會導致音訊透過耳機 (而不是裝置喇叭) 播放。此預設值只能與麥克風搭配使用。

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。 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	<p>高。雙方應該都能發送音樂並在另一側清晰地聽到。</p> <p>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。</p>	<p>高。雙方應該都能發送音樂並在另一側清晰地聽到。</p> <p>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。</p>

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。

注意：使用這些音訊 SDK 方法時，可能會錯誤地設定基礎音訊工作階段。例如，在 iOS 上使用 `.allowBluetooth` 選項搭配 `.playback` 類別會建立無效的音訊組態，而且 SDK 無法錄製或播放音訊。僅在應用程式具有已驗證的特定音訊工作階段需求時使用這些方法。

Android (Kotlin)

```
// This would act similar to the Subscribe Only preset, but it uses a different
// ContentType.
StageAudioManager.getInstance(context)
    .setConfiguration(StageAudioManager.Source.GENERIC,
        StageAudioManager.ContentType.MOVIE,
        StageAudioManager.Usage.MEDIA);

val stage = Stage(context, token, this)

// Other Stage implementation code
```

iOS (Swift)

```
// This would act similar to the Subscribe Only preset, but it uses a different mode
// and options.
IVSStageAudioManager.sharedInstance()
    .setCategory(.playback,
        options: [.duckOthers, .mixWithOthers],
        mode: .default)

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

// Other Stage implementation code
```

iOS 回音消除

iOS 上的回音消除功能也可以使用 `echoCancellationEnabled` 方法透過 `IVSStageAudioManager` 獨立控制。此方法可以控制是否在 SDK 所使用的基礎 `AVAudioEngine` 的輸入和輸出節點上啟用 [語音處理](#)。請務必了解手動變更此屬性所帶來的影響：

- 只有在 SDK 的麥克風處於作用中狀態時，才會執行 `AVAudioEngine` 屬性；這是必要的，因為 iOS 要求同時在輸入和輸出節點啟用語音處理。通常，這是透過使用 `IVSDeviceDiscovery` 傳回的麥克風來建立要發布的 `IVSLocalStageStream` 來完成。或者，可以將 `IVSAudioDeviceStatsCallback` 連接至麥克風本身來啟用麥克風，而不將其用於發布。如果在使用自訂音訊來源型麥克風（而非 IVS SDK 麥克風）時需要回音消除，則此替代方法非常有用。

- 若要啟用 AVAudioEngine 屬性，需要使用 `.videoChat` 或 `.voiceChat` 模式。要求不同的模式會導致 iOS 的基礎音訊架構與 SDK 發生衝突，進而導致音訊遺失。
- 啟用 AVAudioEngine 會自動啟用 `.allowBluetooth` 選項。

行為可能會因裝置和 iOS 版本而異。

iOS 自訂音訊來源

自訂音訊來源可與 SDK 搭配使用，方法是使用 `IVSDeviceDiscovery.createAudioSource`。連線至舞台後，IVS 即時串流廣播 SDK 仍會管理音訊播放的內部 AVAudioEngine 執行個體，即使未使用 SDK 的麥克風也是如此。因此，提供給 `IVSStageAudioManager` 的值必須與自訂音訊來源提供的音訊相容。

如果用於發布的自訂音訊來源是透過麥克風進行錄製，但是由主機應用程式進行管理，除非啟用 SDK 管理的麥克風，否則上述回音消除 SDK 將無法運作。若要繞過該要求，請參閱 [iOS 回音消除](#)。

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 `setEnabledSpeakerphone`，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 參與者錄製狀態變更	錄製開始	發布者已連線至此舞台，且正在將其錄製到 S3。
IVS 參與者錄製狀態變更	錄製結束	發布者已中斷與此舞台的連線，且所有剩餘的檔案都已寫入 S3。
IVS 參與者錄製狀態變更	錄製開始失敗	發布者已連線至此舞台，但由於錯誤 (例如，S3 儲存貯體不存在或不在正確的區域)，錄製無法開始。未錄製此發布者的即時串流
IVS 參與者錄製狀態變更	錄製結束失敗	由於錄製過程中發生錯誤 (例如，嘗試寫入媒體播放清單持續失敗)，錄製以失敗結束。某些物件可能仍會寫入設定的儲存位置。
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。
IVS 階段更新	參與者發布錯誤	參與者嘗試發布至舞台失敗。

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：個別參與者錄製狀態變更

錄製開始：發布者連接至此舞台且被錄製至 S3 時，便會傳送此事件。

```

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

```

錄製結束：發布者中斷與此舞台的連線，且所有剩餘的檔案都已寫入 S3 時，就會傳送此事件。

```

{
  "version": "0",
  "id": "12345678-1a23-4567-a1bc-1a2b34567890",
  "detail-type": "IVS Participant Recording State Change",
  "source": "aws.ivs",
  "account": "123456789012",
  "time": "2024-03-13T22:19:04Z",
  "region": "us-east-1",

```



```

"resources": ["arn:aws:ivs:us-west-2:aws_account_id:stage/AbCdef1G2hij"],
"detail": {
  "session_id": "st-ZyXwvu1T2s",
  "event_name": "Recording End",
  "participant_id": "xYz1c2d3e4f",
  "recording_s3_bucket_name": "bucket-name",
  "recording_s3_key_prefix": "<stage_id>/<session_id>/
<participant_id>/2024-01-01T12-00-55Z"
  "recording_duration_ms": 547327
}
}

```

錄製開始失敗：發布者連接至此舞台，但由於錯誤（例如，S3 儲存貯體不存在或不在正確的區域），錄製無法開始時，就會傳送此事件。發布者的即時串流未被錄製。

```

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

```

錄製結束失敗：由於錄製過程中發生錯誤（例如，嘗試寫入主播放清單失敗），錄製以失敗結束時，就會傳送此事件。某些物件可能仍會寫入設定的儲存位置。

```

{
  "version": "0",
  "id": "12345678-1a23-4567-a1bc-1a2b34567890",
  "detail-type": "IVS Participant Recording State Change",
  "source": "aws.ivs",

```

```

"account": "123456789012",
"time": "2024-03-13T22:19:04Z",
"region": "us-east-1",
"resources": ["arn:aws:ivs:us-west-2:aws_account_id:stage/AbCdef1G2hij"],
"detail": {
  "session_id": "st-ZyXwvu1T2s",
  "event_name": "Recording End Failure",
  "participant_id": "xYz1c2d3e4f",
  "recording_s3_bucket_name": "bucket-name",
  "recording_s3_key_prefix": "<stage_id>/<session_id>/
<participant_id>/2024-01-01T12-00-55Z"
  "recording_duration_ms": 547327
}
}

```

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-ZyXwvu1T2s",
    "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-ZyXwvu1T2s",
    "event_name": "Participant Unpublished",
    "user_id": "Your User Id",
    "participant_id": "xYz1c2d3e4f"
  }
}
```

參與者發布錯誤：參與者嘗試發布至舞台失敗時，就會傳送此事件。

```
{
  "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-ZyXwvu1T2s",
    "event_name": "Participant Publish Error",
    "event_time": "2024-08-13T14:38:17.089061676Z",
    "user_id": "Your User Id",
    "participant_id": "xYz1c2d3e4f",
    "error_code": "BITRATE_EXCEEDED"
  }
}
```

IVS 伺服器端合成 | 即時串流

伺服器端合成使用 IVS 伺服器來混合所有階段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階段主區域中的 IVS 控制平面端點調用。

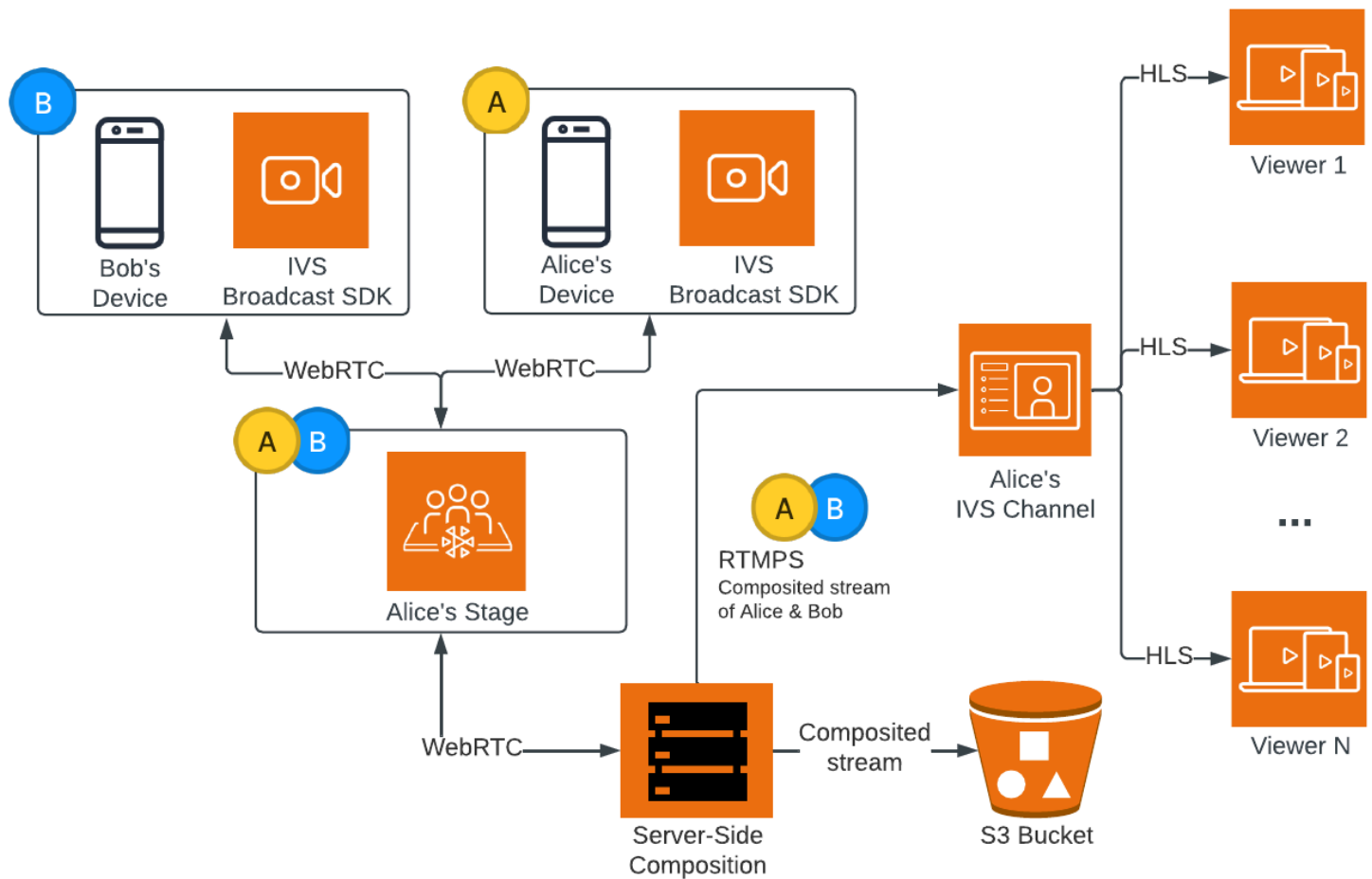
使用伺服器端合成廣播或錄製階段會提供許多好處，可以讓使用者獲得高效可靠的雲端型影片工作流程，使其成為一個吸引人的選擇。

主題

- [IVS 伺服器端合成概觀](#)
- [開始使用 IVS 伺服器端合成](#)
- [在 IVS 伺服器端合成中啟用螢幕共用](#)

IVS 伺服器端合成概觀

下圖說明伺服器端合成的運作方式：



優勢

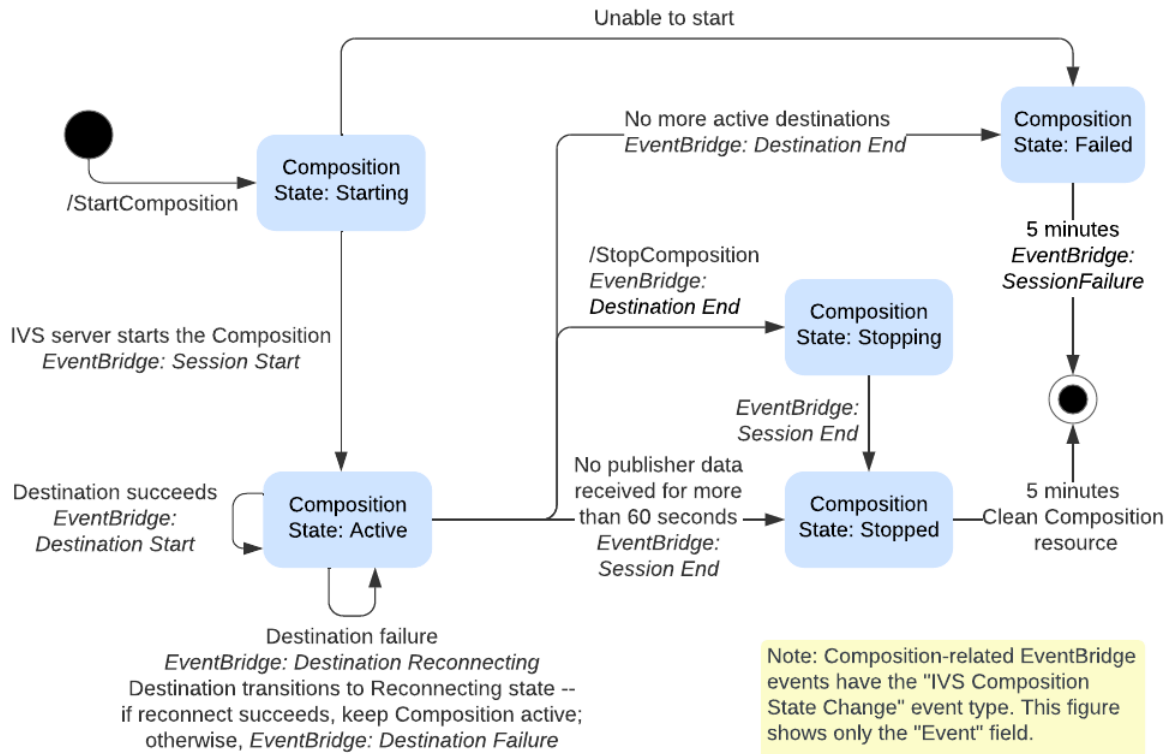
與用戶端合成相比，伺服器端合成具有以下優點：

- **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成員提供更流暢的串流。
- **頻寬效率：**由於伺服器會處理合成，因此階段發布者不必花費額外頻寬將視訊廣播到 IVS。

若要將階段廣播到 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](#)。

合成生命週期

參閱下方圖表，以了解合成的狀態轉換。



合成的生命週期大致如下：

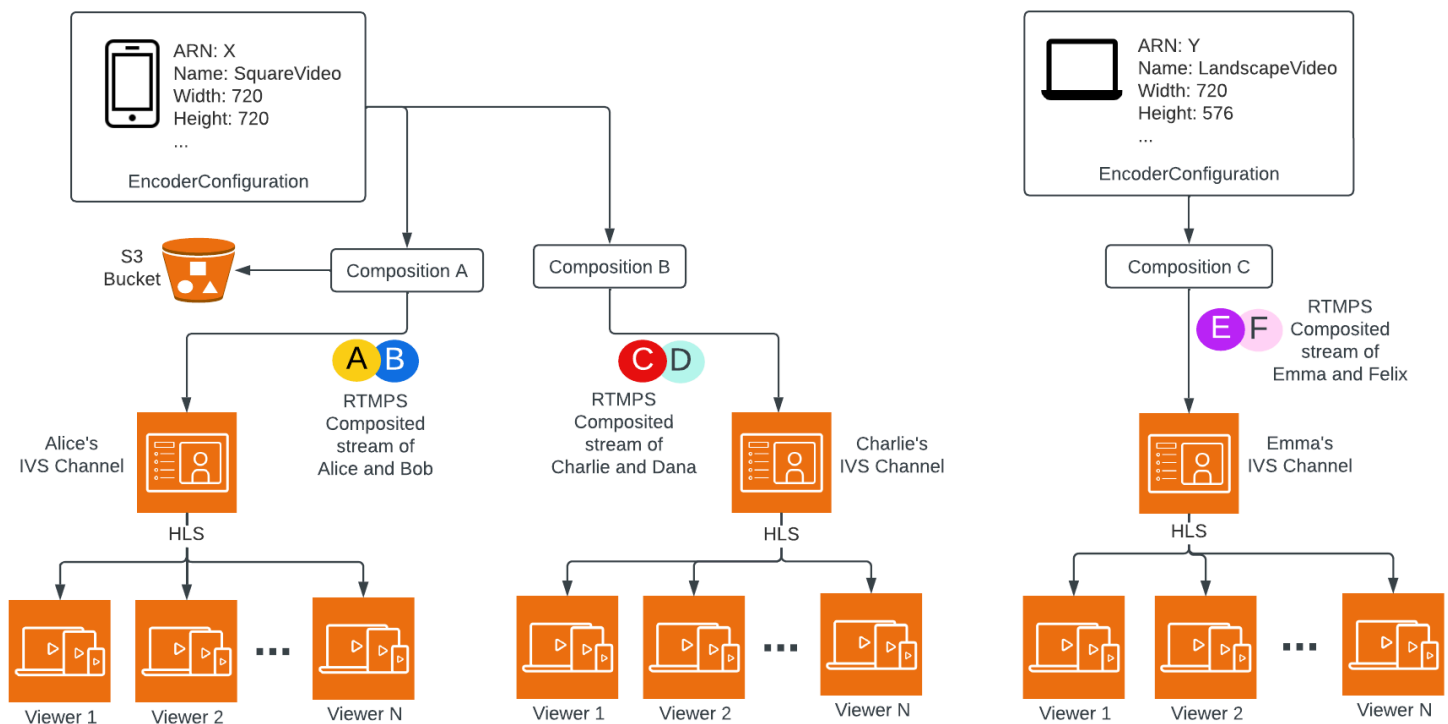
- 當使用者呼叫 `StartComposition` 端點時，會建立合成資源
- 一旦 IVS 成功啟動合成，就會傳送「IVS 合成狀態變更 (工作階段啟動)」EventBridge 事件。如需有關事件的詳細資訊，請參閱[搭配 IVS 即時串流使用 EventBridge](#)。
- 只要合成處於作用中狀態，可能會發生以下情況：
 - 使用者停止合成：如果呼叫 `StopComposition` 端點，IVS 會起始正常關閉合成，並傳送「目的地結束」事件，然後傳送「工作階段結束」事件。
 - 合成執行自動關閉：如果沒有參與者主動發布到 IVS 階段，則合成會在 60 秒後自動完成，並傳送 EventBridge 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。
- 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。)

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

StartComposition 端點提供兩種版面配置選項：網格和 PiP (子母畫面)。

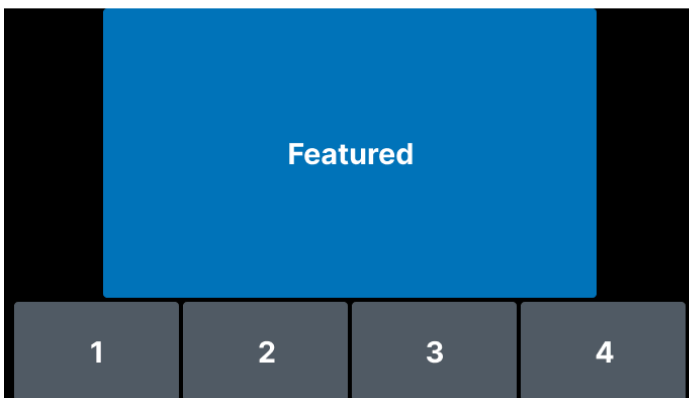
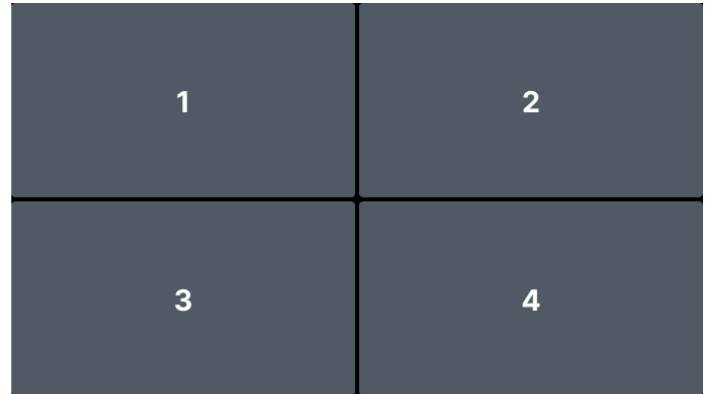
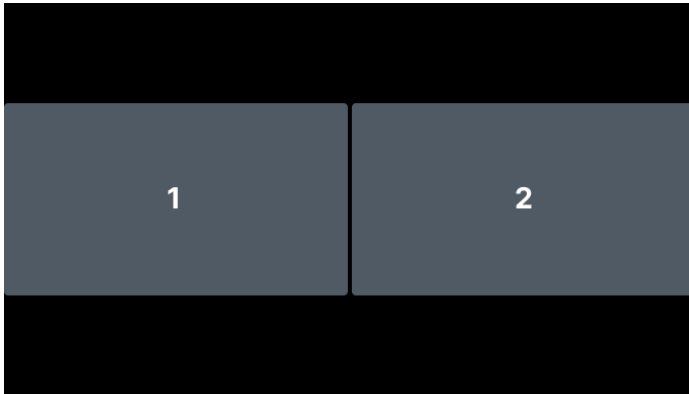
網格版面配置

網格版面配置會將舞台參與者安排在大小相同的網格單元中。此版面配置提供數個可自訂屬性：

- videoAspectRatio 用於設定參與者顯示模式，以控制視訊動態磚的長寬比。

- `videoFillMode` 用於定義影片內容如何適應參與者動態磚。
- `gridGap` 用於指定參與者動態磚之間間距 (以像素為單位)。
- `omitStoppedVideo` 用於允許將已停止的影片串流從合成中排除。
- `featuredParticipantAttribute` 用於識別主播單元。設定此屬性後，主播參與者會顯示在主畫面上較大的單元中，其他參與者顯示在其下方。

如需有關網格版面配置的詳細資訊 (包括所有欄位的有效值和預設值)，請參閱 [GridConfiguration](#) 資料類型。



子母畫面 (PiP) 版面配置

PiP 版面配置可在重疊視窗中顯示參與者，重疊視窗的大小、位置和行為都是可設定的。關鍵屬性包括：

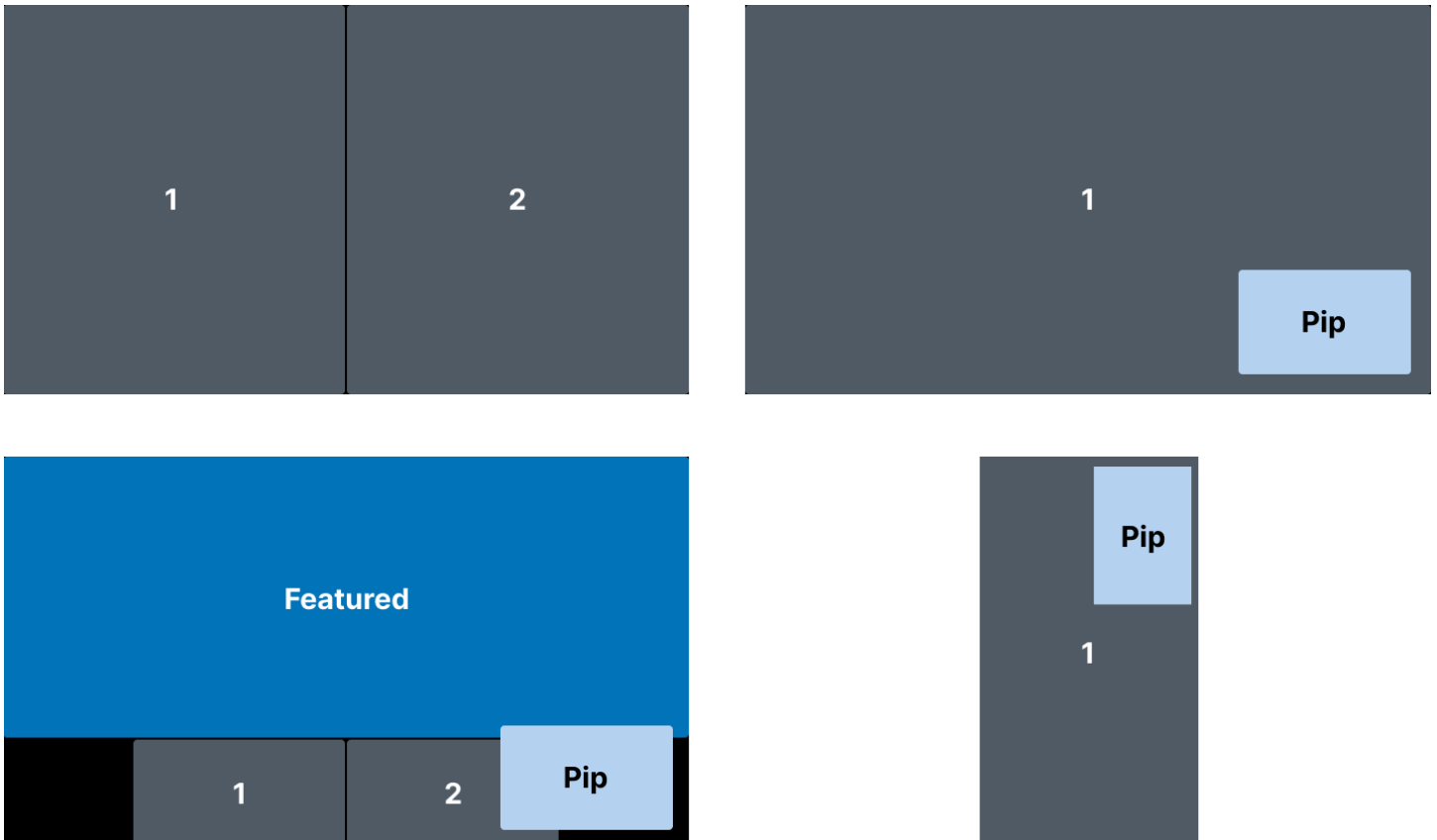
- `pipParticipantAttribute` 用於指定 PiP 視窗的參與者。
- `pipPosition` 用於決定 PiP 視窗的邊角位置。
- `pipWidth` 和 `pipHeight` 用於設定 PiP 視窗的寬度和高度。
- `pipOffset` 用於設定 PiP 視窗相對最近邊緣的偏移位置 (以像素為單位)。

- `pipBehavior` 用於定義所有其他參與者都離開後的 PiP 行為。

與網格版面配置一樣，PiP 支援

`featuredParticipantAttribute`、`omitStoppedVideo`、`videoFillMode` 和 `gridGap` 以供進一步自訂合成。

如需有關 PiP 版面配置的詳細資訊 (包括所有欄位的有效值和預設值)，請參閱 [PipConfiguration](#) 資料類型。



注意：階段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。

重要事項：確保應用程式不依賴目前版面配置的特定功能，例如動態磚的大小和位置。您可以隨時對版面配置進行視覺化改善。

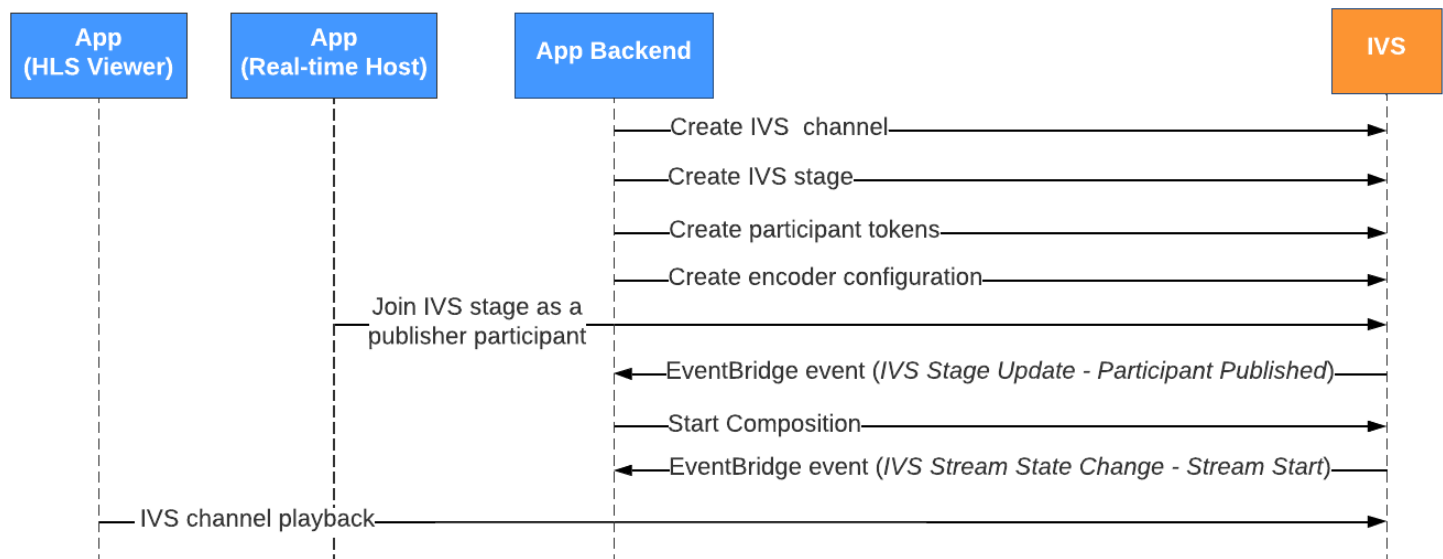
開始使用 IVS 伺服器端合成

本文件將帶您了解開始使用 IVS 伺服器端合成的相關步驟。

必要條件

若要使用伺服器端合成，您必須具有包含作用中發布者的階段，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階段廣播到 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階段直接錄製到 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 階段和參與者權杖。
3. 建立一個 [EncoderConfiguration](#)。
4. 加入並發布到階段。(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. 收到「參與者已發布」EventBridge 事件後，請使用所需的版面配置組態呼叫 [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。

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 命令列介面使用者指南》](#)。

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/9W590BY2M8s4",
    "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：

網格版面配置範例

```
aws ivs-realtime start-composition --stage-arn "arn:aws:ivs:us-
east-1:927810967299:stage/8faHz1SQp0ik" --destinations '[{"channel":
  {"channelArn": "arn:aws:ivs:us-east-1:927810967299:channel/
D01MW4dfMR8r", "encoderConfigurationArn": "arn:aws:ivs:us-
east-1:927810967299:encoder-configuration/9W590BY2M8s4"}}]' --layout '{"grid":
{"featuredParticipantAttribute":"isFeatured","videoFillMode":"COVER","gridGap":0}]'
```

PiP 版面配置範例

```
aws ivs-realtime start-composition --stage-arn "arn:aws:ivs:us-east-1:927810967299:stage/8faHz1SQp0ik" --destinations '[{"channel": {"channelArn": "arn:aws:ivs:us-east-1:927810967299:channel/D0lMW4dfMR8r", "encoderConfigurationArn": "arn:aws:ivs:us-east-1:927810967299:encoder-configuration/DEkQHWPVa0w0"}}]' --layout '{"pip":{"pipParticipantAttribute":"isPip","pipOffset":10,"pipPosition":"TOP_RIGHT"}}'
```

注意：您可以使用[此工具](#)，根據版面配置選擇更輕鬆地產生 --layout 組態。

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 階段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階段，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階段權杖。



```
"arn:aws:ivs:us-east-1:927810967299:encoder-configuration/DEkQHWPVa0w0"}]]]' --layout
'{"grid":{"featuredParticipantAttribute":"screen-share"}}'
```

回應為：

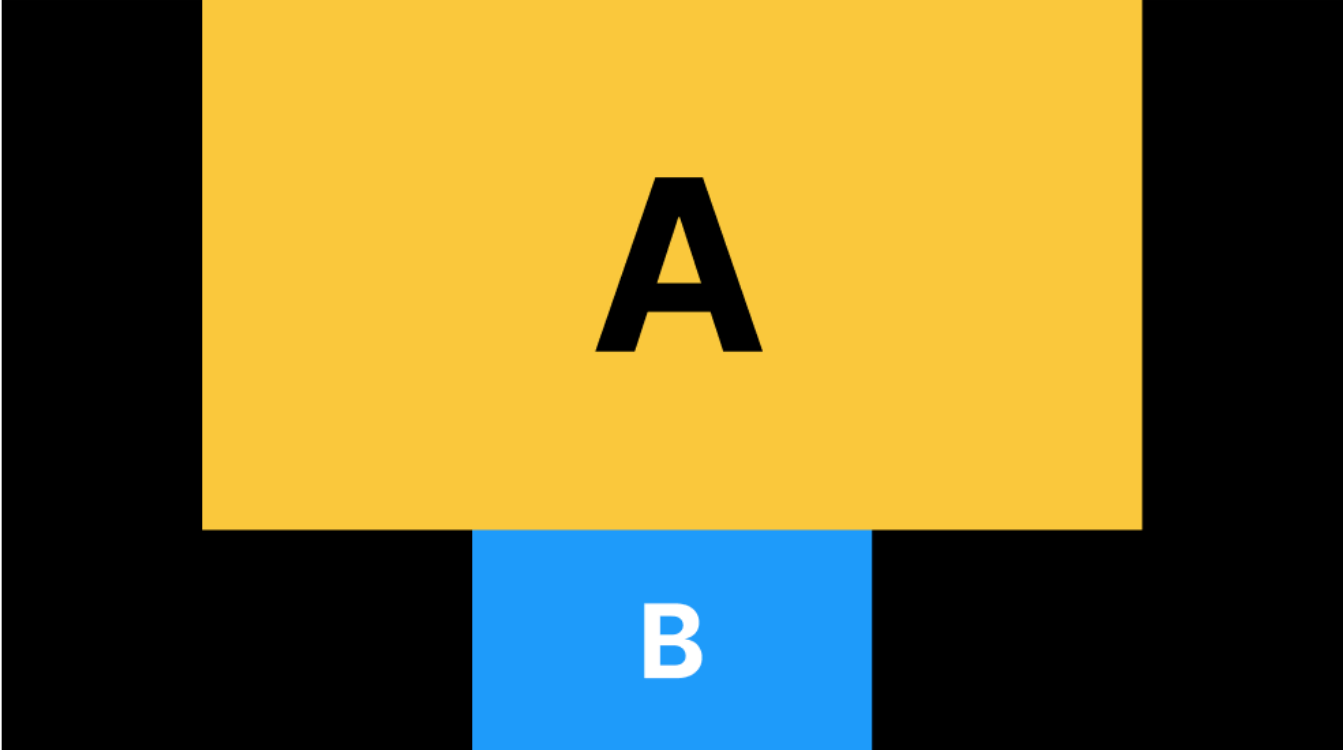
```
{
  "composition" : {
    "arn" : "arn:aws:ivs:us-east-1:927810967299:composition/B19tQcXRgtoz",
    "destinations" : [ {
      "configuration" : {
        "channel" : {
          "channelArn" : "arn:aws:ivs:us-east-1:927810967299:channel/D01MW4dfMR8r",
          "encoderConfigurationArn" : "arn:aws:ivs:us-east-1:927810967299:encoder-configuration/DEkQHWPVa0w0"
        },
        "name" : ""
      },
      "id" : "SGmgBXTULuXv",
      "state" : "STARTING"
    } ],
    "layout" : {
      "grid" : {
        "featuredParticipantAttribute" : "screen-share",
        "gridGap": 2,
        "omitStoppedVideo": false,
        "videoAspectRatio": "VIDEO"
      }
    },
    "stageArn" : "arn:aws:ivs:us-east-1:927810967299:stage/8faHz1SQp0ik",
    "startTime" : "2023-09-27T21:32:38Z",
    "state" : "STARTING",
    "tags" : { }
  }
}
```

當階段參與者 E813MFk1PWLF 加入階段時，該參與者的視訊將顯示在精選插槽中，其餘階段發布者將會在插槽下方轉譯：

Channel details

Channel name test-channel	Channel type Standard	Video latency Low
Playback authorization Disabled	Auto-record to S3 Disabled	ARN 

▼ Live stream



Note: Playback will consume resources, and you will incur live video output cost. [Learn more](#)

State LIVE	Health ✔ Healthy	Duration 00:00:08	Viewers 0
----------------------	---------------------	----------------------	--------------

▶ Timed Metadata

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
```


IVS 錄製 | 即時串流

IVS 即時串流有兩種錄製選項：

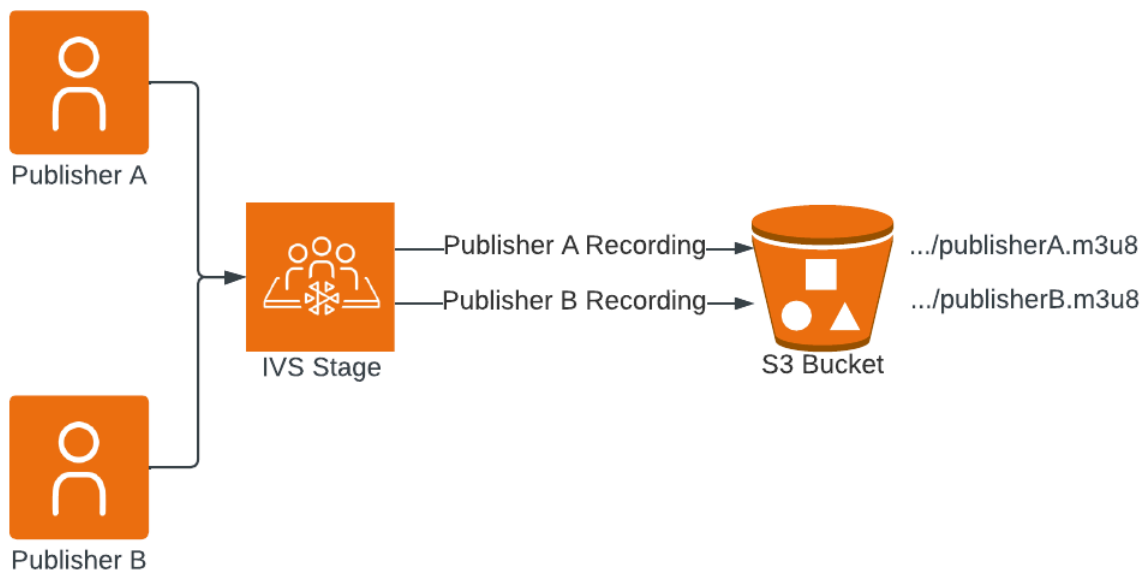
- 個別參與者錄製會將每個發布者的媒體錄製在個別檔案中。
- 但是，複合錄製會將來自所有發布者的媒體合併為單一檢視，錄製在單一檔案中。

個別參與者錄製不會產生額外的 Amazon IVS 費用，但複合錄製會按每小時費率產生影片編碼費用。這兩種錄製選項都會產生標準 S3 儲存和請求費用。如需更多詳細資訊，請參閱 [Amazon IVS 定價](#)。

如需自訂程度更高的解決方案，請考慮使用開放原始碼 [IVSStageSaver](#) 專案，作為您自己自主託管錄製服務的基礎。

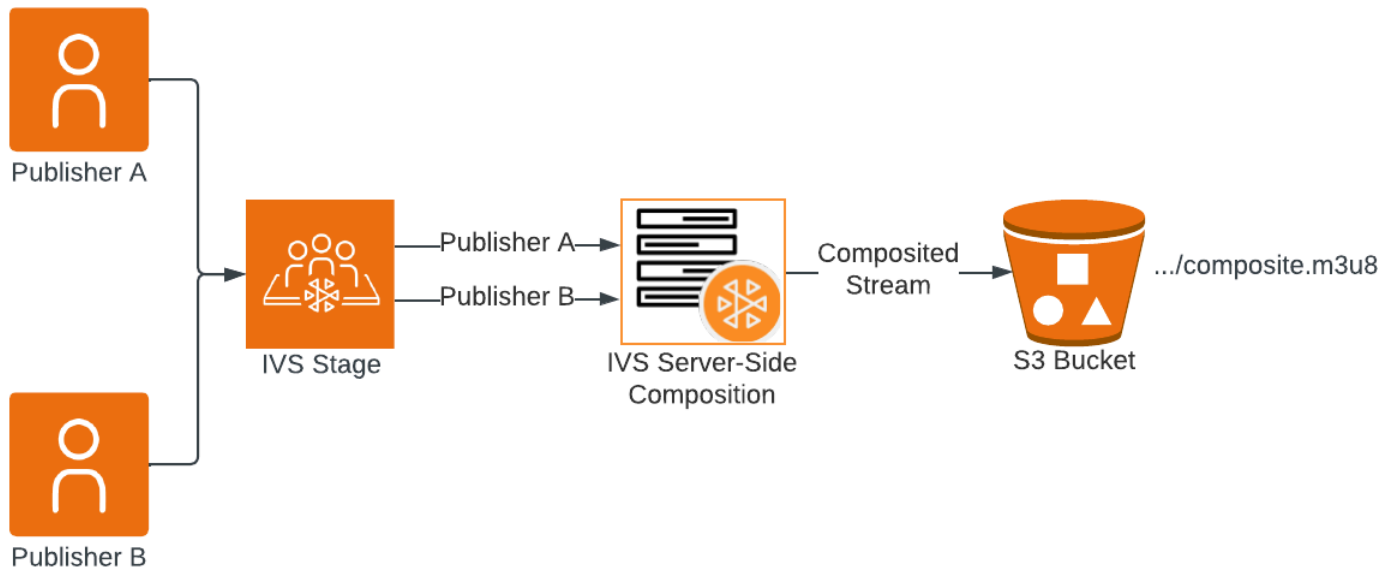
個別參與者錄製

此選項非常適合具有單一發布者的即時串流，或需要對每個發布者進行個別錄製時 (特別是用於審核目的時)。如需更多詳細資訊，請參閱 [個別參與者錄製](#)。



複合錄製

此選項會將來自多個發布者的媒體合併為單一檢視並錄製在單一檔案中，非常適合隨選視訊體驗。如需更多詳細資訊，請參閱 [複合錄製](#)。



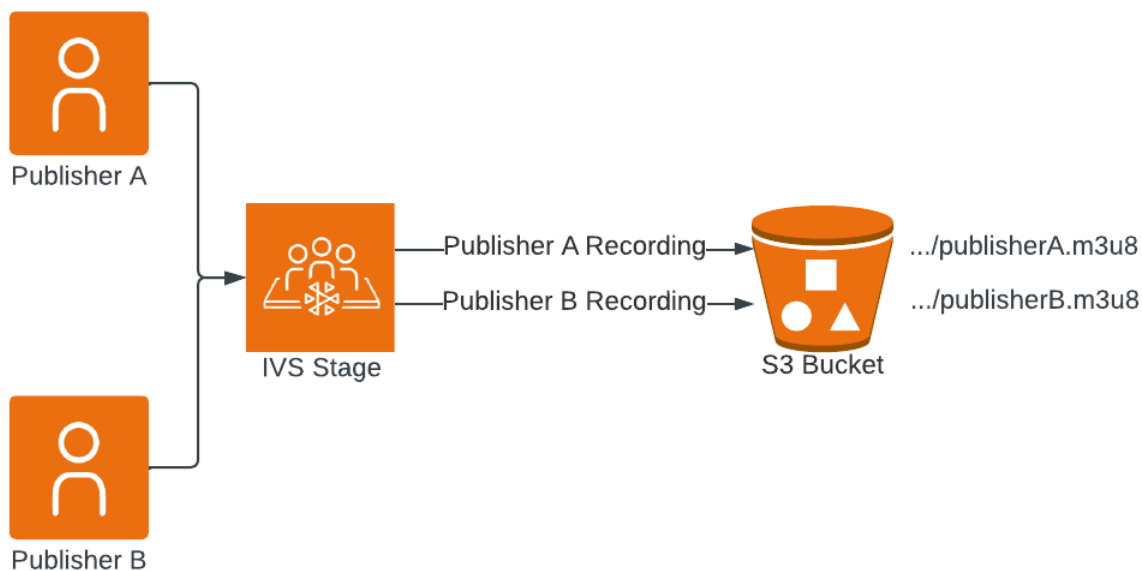
IVS 個別參與者錄製 | 即時串流

本文件說明如何搭配 IVS 即時串流使用個別參與者錄製功能。

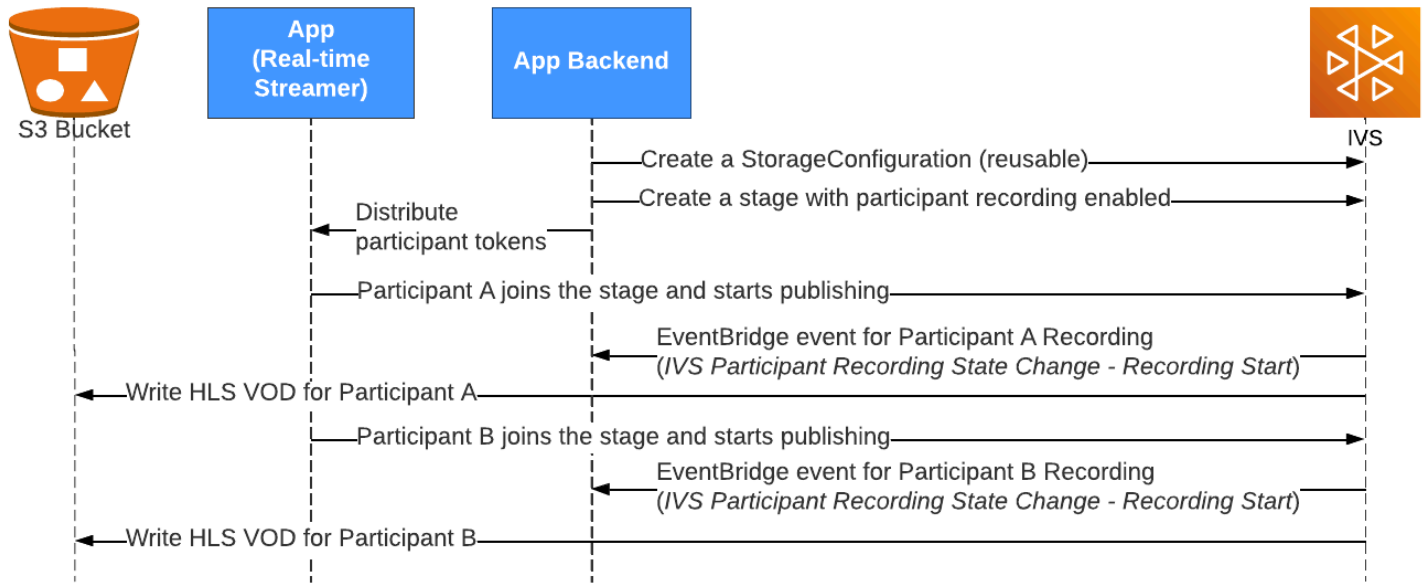
簡介

個別參與者錄製允許 IVS 即時串流客戶，將 IVS 舞台發布者個別錄製到 S3 儲存貯體。啟用舞台的個別參與者錄製後，發布者開始向舞台發布內容後，就會錄製發布者內容。

注意：如果您要將所有舞台參與者混入一段影片中，則複合錄製功能會更為合適。如需錄製 IVS 即時串流內容的摘要，請參閱[錄製](#)。



工作流程



1. 建立 S3 儲存貯體

您需要 S3 儲存貯體來寫入 VOD。如需詳細資訊，請參閱 S3 文件：[如何建立儲存貯體](#)。請注意，對於個別參與者錄製，必須在與 IVS 舞台相同的 AWS 區域中建立 S3 儲存貯體。

重要事項：如果您使用現有的 S3 儲存貯體，則物件擁有權設定必須是強制執行的儲存貯體擁有者，或是偏好的儲存貯體擁有者。如需詳細資訊，請參閱 S3 文件：[控制物件擁有權](#)。

2. 建立 StorageConfiguration 物件

建立儲存貯體後呼叫 IVS 即時串流 API，以[建立 StorageConfiguration](#) 物件。成功建立儲存體組態後，IVS 將獲得許可，可寫入所提供 S3 儲存貯體。您可以將此 StorageConfiguration 物件重複用於多個舞台。

3. 建立具有參與者權杖的舞台

現在，您需要[建立 IVS 舞台](#)，並為舞台啟用個別參與者錄製 (方式是設定 AutoParticipantRecordingConfiguration 物件)，同時為每個發布者建立參與者權杖。

以下請求會建立一個具有兩個參與者權杖並啟用個別參與者錄製的舞台。

```
POST /CreateStage HTTP/1.1
```

```
Content-type: application/json

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

4. 以作用中發布者身分加入此舞台

將參與者權杖分發給發布者，讓他們加入此舞台並開始[向舞台發布內容](#)。

發布者加入此舞台並開始使用其中一個 [IVS 即時串流廣播 SDK](#) 向舞台發布內容時，參與者錄製程序會自動開始，並向您傳送 [EventBridge 事件](#)，指出錄製已開始。(此事件為「IVS 參與者錄製狀態變更 – 錄製開始」。) 同時，參與者錄製程序會開始將 VOD 和中繼資料檔案寫入設定的 S3 儲存貯體。注意：不保證會將連線持續時間極短 (少於 5 秒) 的參與者錄製下來。

有兩種方式可以取得每個錄製的 S3 字首：

- 接聽 EventBridge 事件：

```
{
  "version": "0",
  "id": "12345678-1a23-4567-a1bc-1a2b34567890",
  "detail-type": "IVS Participant Recording State Change",
  "source": "aws.ivs",
  "account": "123456789012",
  "time": "2024-03-13T22:19:04Z",
```

```

"region": "us-east-1",
"resources": ["arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij"],
"detail": {
  "session_id": "st-ZyXwvu1T2s",
  "event_name": "Recording Start",
  "participant_id": "xYz1c2d3e4f",
  "recording_s3_bucket_name": "ivs-recordings",
  "recording_s3_key_prefix": "<stage_id>/<session_id>/
<participant_id>/2024-01-01T12-00-55Z"
}
}

```

- 使用 [GetParticipant](#) API 端點 – 此回應包含指向參與者錄製檔案位置的 S3 儲存貯體和字首。以下是請求內容：

```

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

```

以下是回應內容：

```

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

```

5. 播放 VOD

錄製結束後，您可以使用 [IVS 播放器](#) 觀看內容。如需有關如何設定 CloudFront 分佈以進行 VOD 播放的說明，請參閱 [從私有儲存貯體播放錄製的內容](#)。

純音訊錄製

設定個別參與者錄製時，可以選擇將純音訊 HLS 區段寫入 S3 儲存貯體。若要使用此功能，請在建立舞台時選擇 AUDIO_ONLY mediaType：

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

{
  "autoParticipantRecordingConfiguration": {
    "storageConfigurationArn": "arn:aws:ivs:us-west-2:123456789012:storage-configuration/AbCdef1G2hij",
    "mediaTypes": ["AUDIO_ONLY"]
  },
  "name": "TestStage",
  "participantTokenConfigurations": [
    {
      "capabilities": ["PUBLISH", "SUBSCRIBE"],
      "duration": 20160,
      "userId": "1"
    },
    {
      "capabilities": ["PUBLISH", "SUBSCRIBE"],
      "duration": 20160,
      "userId": "2"
    }
  ]
}
```

錄製內容

當個別參與者錄製處於作用中狀態時，您將開始看到 HLS 影片區段和中繼資料檔案寫入舞台建立時提供的 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，錄製結束後，「IVS 參與者錄製狀態會變更 – 錄製結束」事件會透過 EventBridge 傳送。建議您只在接收此事件之後播放或處理錄製的串流。如需詳細資訊，請參閱[搭配 IVS 即時串流使用 EventBridge](#)。

以下是即時 IVS 工作階段錄製的範例目錄結構和內容：

```
s3://mybucket/stageId/stageSessionId/participantId/timestamp
events
```

```

recording-started.json
recording-ended.json
media
  hls
multivariant.m3u8
  high
    playlist.m3u8
    1.mp4

```

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 子資料夾包含錄製工作階段期間產生的所有媒體和資訊清單檔案，而且此子資料夾可以使用 IVS 播放器播放

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	說明
stage_arn	string	是	錄製來源舞台 ARN。
session_id	string	是	字串，表示錄製的參與者所在舞台的 session_id。
participant_id	string	是	字串，表示錄製的參與者識別碼。
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_FA

欄位	類型	必要	說明
			ILED，則無法使用此選項。此外，請參閱下文 recording_ended_at 中的備註。
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。 注意：recording_started_at 和 recording_ended_at 是這些事件產生時的時間戳記，未必與 HLS 影片區段時間戳記完全一致。若要準確判斷錄製的持續時間，請使用 duration_ms 欄位。
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。
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。

欄位	類型	必要	說明
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。
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。
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:us-west-2:aws_account_id:stage/AbCdef1G2hij",
  "session_id": "st-ZyXwvu1T2s",
  "participant_id": "xYz1c2d3e4f",
  "recording_started_at": "2024-03-13T13:17:17Z",
  "recording_status": "RECORDING_STARTED",
  "media": {
    "hls": {
      "path": "media/hls",
      "playlist": "multivariant.m3u8",
      "renditions": [
        {
```

```
        "path": "high",
        "playlist": "playlist.m3u8"
      }
    ]
  }
}
```

範例 : recording-ended.json

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

範例 : recording-failed.json

```
{
  "version": "v1",
  "stage_arn": "arn:aws:ivs:us-west-2:aws_account_id:stage/AbCdef1G2hij",
  "session_id": "st-ZyXwvu1T2s",
  "participant_id": "xYz1c2d3e4f",
  "recording_started_at": "2024-03-13T19:44:19Z",
  "recording_ended_at": "2024-03-13T19:55:04Z",
```

```
"recording_status": "RECORDING_ENDED_WITH_FAILURE",
"media": {
  "hls": {
    "duration_ms": 645237,
    "path": "media/hls",
    "playlist": "multivariant.m3u8",
    "renditions": [
      {
        "path": "high",
        "playlist": "playlist.m3u8"
      }
    ]
  }
}
```

IVS 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。

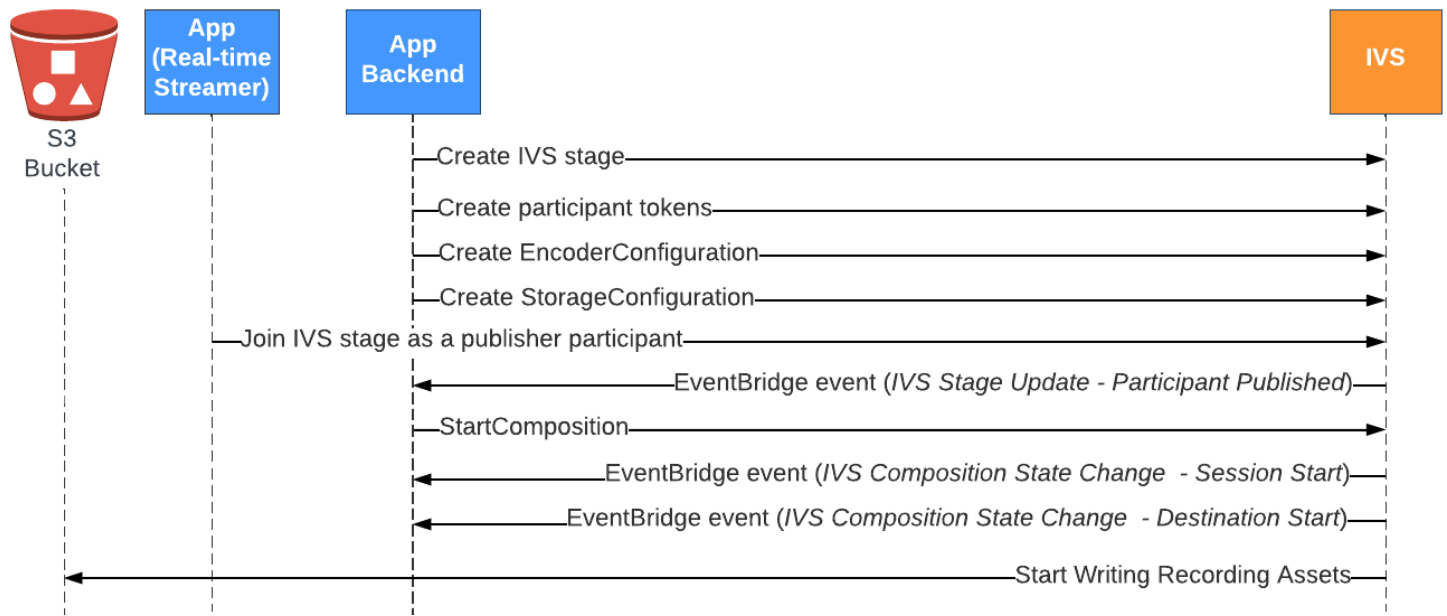
必要條件

若要使用複合錄製，您必須具有包含作用中發布者的舞台，並使用 S3 儲存貯體作為錄製目的地。下面我們描述了一種使用 EventBridge 事件將合成錄製到 S3 儲存貯體的可能工作流程。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. 建立一個 [EncoderConfiguration](#) (此物件表示應如何轉譯錄製的影片)。
3. 建立 [S3 儲存貯體](#)和 [StorageConfiguration](#) (錄製內容的儲存位置)。

重要事項：如果您使用現有的 S3 儲存貯體，則物件擁有權設定必須是強制執行的儲存貯體擁有者，或是偏好的儲存貯體擁有者。如需詳細資訊，請參閱 S3 文件：[控制物件擁有權](#)。

4. [加入並發布到階段](#)。
5. 當您收到參與者已發布的 [EventBridge 事件](#)時，請呼叫以 S3 DestinationConfiguration 物件為目的地的 [StartComposition](#)
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：以 S3 儲存貯體為目的地的 StartComposition

下列範例顯示對 [StartComposition](#) 端點的典型呼叫，並將 S3 指定為合成的唯一目的地。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/ZBcEbgE24Cq"
      }
    }
  ]
}
  
```

```

    }
  }
],
"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/"
          }
        },
        "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": {}
  }
}

```

```
}  
}
```

出現在 StartComposition 回應中的 recordingPrefix 欄位可用來決定複製內容的儲存位置。

錄製內容

當合成轉換到 ACTIVE 狀態時，您將開始看到 HLS 影片片段和中繼資料檔案寫入到呼叫 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[搭配 IVS 即時串流使用 EventBridge](#)。

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。

StorageConfiguration 的儲存貯體政策

建立 StorageConfiguration 物件時，IVS 會取得將內容寫入指定 S3 儲存貯體的存取權。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。

欄位	類型	必要	描述
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。
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。

欄位	類型	必要	描述
recording_ended_at	string	有條件	<p>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。</p> <p>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。</p>
recording_started_at	string	有條件	<p>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。</p> <p>請參閱上面的 recording_ended_at 備註。</p>
recording_status	string	是	<p>錄製的狀態。有效值："RECORDING_STARTED"、"RECORDING_ENDED"、"RECORDING_START_FAILED"、"RECORDING_ENDED_WITH_FAILURE"。</p>
recording_status_message	string	有條件	<p>狀態的描述性資訊。當 recording_status 為 "RECORDING_ENDED" 或 "RECORDING_ENDED_WITH_FAILURE" 時才可用。</p>
version	string	是	<p>中繼資料結構描述的版本。</p>

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」)。不過，您可以使用 Amazon CloudFront CDN (內容交付網路) 播放這些檔案。

CloudFront 分佈可設定為從私有儲存貯體提供內容。普遍情況下，這比擁有可開放存取的儲存貯體來繞過 CloudFront 提供的控制項更好。您可以設定分佈，透過建立原始存取控制 (OAC) 從私有儲存貯體提供服務；該身分是對私有來源儲存貯體具有讀取許可的特殊 CloudFront 使用者。您可以在建立分

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[建立新的原始存取控制](#)。

在啟用 CORS 的情況下使用 CloudFront 設定播放

此範例說明開發人員如何在啟用 CORS 的情況下設定 CloudFront 分佈，以便從任何網域播放錄製檔。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 分佈

請參閱《CloudFront 開發人員指南》中的[建立 CloudFront 分佈](#)。

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
Web 應用程式防火牆	啟用安全保護

然後儲存 CloudFront 分佈。

步驟 3：設定 S3 儲存貯體政策

1. 刪除您為 S3 儲存貯體設定的任何 StorageConfiguration。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. 前往 CloudFront 分佈，確保所有分佈欄位都處於上一步中定義的狀態，然後複製儲存貯體政策 (使用複製政策按鈕)。
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，儲存貯體政策中應該只有 CloudFront 政策。
4. 建立 StorageConfiguration，指定 S3 儲存貯體。

建立 StorageConfiguration 之後，您會在 S3 儲存貯體政策中看到兩個項目，一個允許 CloudFront 讀取內容，另一個允許 IVS 寫入內容。在[範例：具備 CloudFront 和 IVS 存取權的 S3 儲存貯體政策](#)部分中，顯示了具有 CloudFront 和 IVS 存取權的最終儲存貯體政策範例。

步驟 4：播放錄製檔

成功設定 CloudFront 分佈並更新儲存貯體政策之後，您應該可以使用 IVS 播放器播放錄製檔：

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. 按照此範例中的步驟 1 到步驟 3 操作之後，您應該可以透過 CloudFront URL 使用影片檔案。您的 CloudFront URL 是 Amazon CloudFront 主控台中詳細資訊索引標籤上的分佈網域名稱。外觀大致如下：

```
a1b23cdef4ghij.cloudfront.net
```

3. 若要透過 CloudFront 分佈播放錄製的影片，請在 S3 儲存貯體下尋找 `multivariant.m3u8` 檔案的物件金鑰。外觀大致如下：

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

4. 將物件金鑰附加到 CloudFront URL 結尾。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：具備 CloudFront 和 IVS 存取權的 S3 儲存貯體政策

下面的程式碼片段描繪了一段 S3 儲存貯體政策，該政策允許 CloudFront 將內容讀取到私有儲存貯體，允許 IVS 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。您的政策應包含與自己 CloudFront 分佈和 StorageConfiguration 相關的 ID。

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "CompositeWrite-7eiKaIGkC9D0",  
      "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

- 合成沒被寫入 S3 儲存貯體：請確保已在相同區域中建立 S3 儲存貯體和 StorageConfiguration 物件。另外，請檢查儲存貯體政策，確保 IVS 能夠存取儲存貯體；請參閱 [StorageConfiguration 的儲存貯體政策](#)。
- 我在執行 ListCompositions 時找不到合成：合成是短暫的資源。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。

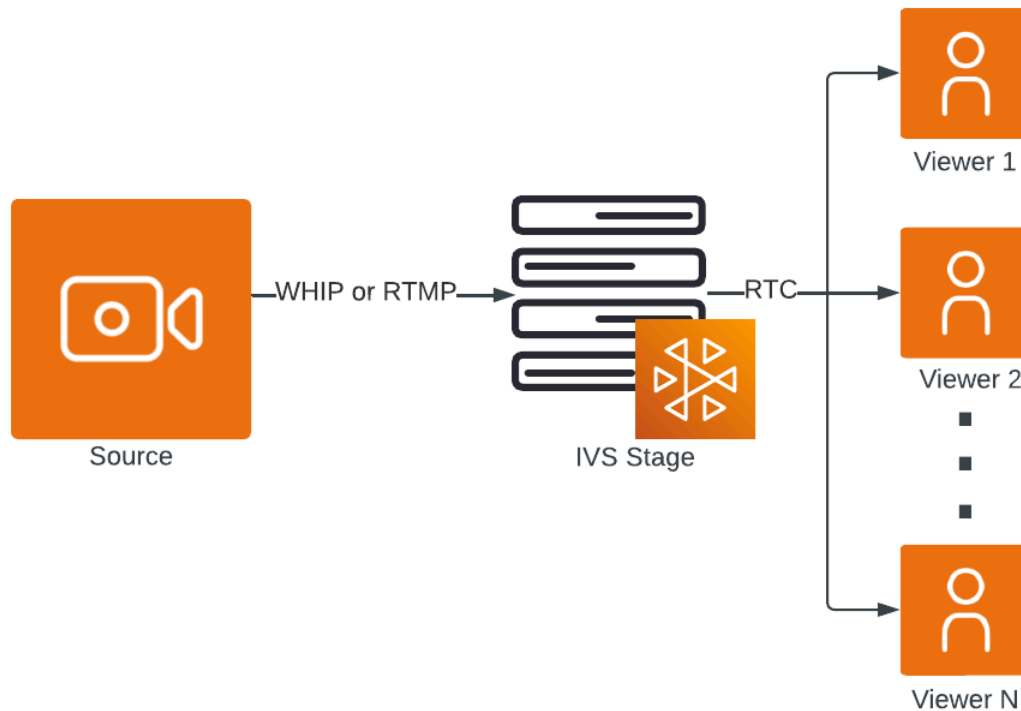
已知問題

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。

IVS 串流擷取 | 即時串流

除了使用 IVS 廣播 SDK 之外，您也可以從 WHIP 或 RTMP 來源將影片發布至 IVS 舞台。如果使用 SDK 不可行或不是偏好的方式 (例如從 OBS Studio 或硬體編碼器發布影片時)，此方法就能為您提供工作流程靈活性。建議您盡可能使用 IVS 廣播 SDK，因為我們無法保證第三方解決方案與 IVS 的相容性或是搭配使用時的效能。

此圖表說明使用 WHIP 和 RTMP 進行發布的運作方式：



支援的通訊協定

IVS 即時串流支援數個擷取通訊協定：

- RTMP (即時訊息通訊協定) – 是透過網路傳輸視訊的業界標準。
- RTMPS – 透過 TLS 運作的安全 RTMP 版本。
- WHIP (WebRTC-HTTP 擷取通訊協定) – 為標準化 WebRTC 擷取而開發的 IETF 草案。

通常，RTMP 的延遲比 WHIP 更高，因此非常適合一對多的即時串流。如需有關使用這些通訊協定的詳細指引，請參閱 [RTMP](#) 和 [WHIP](#) 文件。

支援的媒體規格

- 音訊輸入格式
 - 編解碼器：AAC-LC (用於 RTMP) 和 Opus (用於 WHIP)
 - 聲道：2 (立體聲) 或 1 (單聲道)
 - 取樣率：44.1 kHz 或 48 kHz
 - 位元速率上限：160 Kbps
- 視訊輸入格式
 - 編解碼器：H.264
 - H.264 設定檔：基準
 - IDR 間隔：1 或 2 秒
 - 影格速率：10 到 60 FPS
 - B 影格：0

注意：使用 RTMP 時，IVS 廣播 SDK 預設會啟用 B 影格。因此，開發人員必須停用 B 影格：在 iOS 上使用 `usesBFrames` 方法；在 Android 上使用 `setUseBFrames`。如果開發人員未停用 B 影格，串流將會遭到中斷。

- 解析度：上限：720p。下限：160p
- 位元速率上限：8.5 Mbps
- 編碼器組態：建議您使用 H.264 編碼器的 `veryfast` 和 `zerolatency` 設定。此外：`zerolatency` 預設中包含 `sliced_threads x264` 選項，建議您停用該選項。例如，使用 FFmpeg 時，命令應包含：`-preset:v veryfast -tune zerolatency -x264-params sliced-threads=0`

IVS RTMP 發布 | 即時串流

本文件概述使用 RTMP 發布至 IVS 舞台的程序。如需有關各種擷取選項的其他詳細資訊，請參閱[串流擷取文件](#)

建立舞台

若要建立舞台，請使用下列命令：

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

如需詳細資訊 (包括回應)，請參閱 [CreateStage](#)。

重要事項：記下回應中的 endpoints 欄位，其中同時列出 RTMP 和 RTMPS 端點。這些是設定 RTMP 編碼器的必要內容。

建立擷取組態

若要使用 RTMP 發布至舞台，您必須先建立擷取組態，並將其與舞台建立關聯。發布至此舞台 (使用擷取組態中的串流金鑰和舞台的 RTMP 端點) 時，媒體會作為參與者發布至舞台。您可以選擇指定 `userId` 和自訂 `attributes`，這會與連線至此舞台的 [參與者](#) 建立關聯。

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

如需詳細資訊 (包括回應)，請參閱 [CreateIngestConfiguration](#)。

建立擷取組態時，您可以預先將其與特定舞台 ARN 建立關聯。如果沒有此關聯，串流金鑰就無法使用。此外，可以透過 [UpdateIngestConfiguration](#) 端點更新擷取組態 (包括 `stageArn` 欄位)，將相同的組態重複用於不同的舞台。

注意：擷取組態 `insecureIngest` 欄位預設為 `false`，需要使用 RTMPS。RTMP 連線將遭到拒絕。如果您必須使用 RTMP，請將 `insecureIngest` 設為 `true`。除非您有需使用 RTMP 的經驗證特定使用案例，否則我們建議您使用 RTMPS。

使用 RTMP 編碼器發布

此範例示範如何使用 OBS Studio；不過，您可以使用符合 IVS [媒體規格](#) 的任何 RTMP 編碼器。

1. 下載並安裝軟體：<https://obsproject.com/download>。
2. 按一下 設定。在設定面板的串流區段中，從服務下拉式清單中選取自訂。
3. 在伺服器欄位中，輸入此舞台的 RTMP 或 RTMPS 端點。
4. 在串流金鑰欄位中，輸入擷取組態的 `streamKey`。
5. 如往常一樣進行影片設定，但需要遵守一些限制：
6. a. IVS 即時串流支援在 8.5 Mbps 高達 720p 的輸入。如果超過上述任一限制，串流將會遭到中斷。
b. 建議您在輸出面板中，將關鍵影格間隔設定為 1 秒或 2 秒。縮短關鍵影格間隔可更快地為觀眾開始播放影片。我們也建議將 CPU 使用量預設值設為非常快速，並將調校設為零延遲，以實現最低延遲。

- c. 由於 OBS 不支援 Simulcast，因此建議您將位元速率保持在 2.5 Mbps 以下。這可讓使用低頻寬連線的觀眾也能觀看。
- d. 停用 B 影格，因為具有 B 影格的串流會自動遭到中斷。執行以下任意一項：
 - 在 x264 選項中，輸入 `bframes=0 sliced-threads=0`。
 - 如果 B 影格選項可用 (例如用於 NVENC 時)，則將其設定為 0。

7. 選取開始串流

重要事項：如果將編碼器的最大位元速率設定為 8.5 Mbps，則發布者偶爾會從工作階段中消失。這是因為最大位元速率設定只是目標，編碼器偶爾會超過此目標。若要防止這種情況，請將編碼器的最大位元速率設為較低的值；例如，設為 6 Mbps。

IVS WHIP 發布 | 即時串流

本文件說明如何使用 OBS 等 WHIP 相容編碼器，來發布至 IVS 即時串流。[WHIP](#) (WebRTC-HTTP 擷取通訊協定) 是為了標準化 WebRTC 擷取而開發的 IETF 草案。

WHIP 可以與 OBS 等軟體相容，為桌面發布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。這為開發人員提供多元的選項：使用 IVS Web 廣播 SDK 直接在瀏覽器進行發布，或允許實況主在其桌面使用 OBS 以存取更強大的工具。

此外，在不可使用或不偏好使用 IVS 廣播 SDK 的情形下，WHIP 就很有用。例如，在涉及硬體編碼器的設定中，IVS 廣播 SDK 可能不適用。但是，如果編碼器支援 WHIP，您仍然可以直接從編碼器發布到 IVS。

OBS 指南

自第 30 版起，OBS 支援 WHIP。若要開始，請下載 OBS 第 30 版或更新版本：<https://obsproject.com/>。

若要透過 WHIP 使用 OBS 發布至 IVS 舞台，請遵循下列步驟：

1. [產生](#)具有發布功能的參與者權杖。就 WHIP 而言，參與者權杖是持有人權杖。根據預設，參與者權杖會在 12 小時後過期，但您可以將持續時間延長至 14 天。
2. 按一下 設定。在設定面板的串流區段中，從服務下拉式清單中選取 WHIP。
3. 在伺服器欄位中，輸入 `https://global.whip.live-video.net`。
4. 在持有人權杖欄位中，輸入您在步驟 2 中產生的參與者權杖。

5. 如往常一樣進行影片設定，但需要遵守一些限制：
 - a. IVS 即時串流支援在 8.5 Mbps 高達 720p 的輸入。如果超過上述任一限制，串流將會遭到中斷。
 - b. 建議您在輸出面板中，將關鍵影格間隔設定為 1 秒或 2 秒。縮短關鍵影格間隔可更快地為觀眾開始播放影片。我們也建議將 CPU 使用量預設值設為非常快速，並將調校設為零延遲，以實現最低延遲。
 - c. 由於 OBS 不支援 Simulcast，因此建議您將位元速率保持在 2.5 Mbps 以下。這可讓使用低頻寬連線的觀眾也能觀看。
6. 按下開始串流。

注意：我們留意到 OBS 中可能發生的與 WHIP 相關的品質問題 (例如間歇性影片凍結)。這些問題通常會在廣播者的網路不穩定時發生。建議您在 OBS 中測試 WHIP，然後再將其用於生產即時串流。降低廣播位元速率也有助於減少這些問題的發生次數。

IVS 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
IngestConfiguration	CreateIngestConfiguration	5 TPS
IngestConfiguration	DeleteIngestConfiguration	5 TPS
IngestConfiguration	GetIngestConfiguration	5 TPS
IngestConfiguration	ListIngestConfigurations	5 TPS
IngestConfiguration	UpdateIngestConfiguration	5 TPS
MediaEncoder	CreateEncoderConfiguration	5 TPS

端點類型	端點	預設
MediaEncoder	DeleteEncoderConfiguration	5 TPS
MediaEncoder	GetEncoderConfiguration	5 TPS
MediaEncoder	ListEncoderConfigurations	5 TPS
PublicKey	DeletePublicKey	3 TPS
PublicKey	GetPublicKey	3 TPS
PublicKey	ImportPublicKey	3 TPS
PublicKey	ListPublicKeys	3 TPS
階段	CreateParticipantToken	50 TPS
階段	CreateStage	5 TPS
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

端點類型	端點	預設
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	每個帳戶的 EncoderConfiguration 物件數量上限。
合成目的地	2	否	合成物件中目的地物件數量上限。
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	每個帳戶的並行合成物件數量上限。
IngestConfigurations	100	是	每個帳戶的 IngestConfiguration 物件數量上限。
參與者發布位元速率	8.5 Mbps	否	可以串流至舞台的每秒位元數上限。

資源或功能	預設	可調整	描述
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。
PublicKeys	3	否	每個 AWS 區域的公有金鑰數量上限。
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 (最多 25,000)	一次可以訂閱階段的最大參與者人數。
階段	1,000	是	每個 AWS 區域的階段數目上限。
StorageConfigurations	5	是	每個帳戶的 StorageConfiguration 物件數量上限。

IVS 即時串流最佳化

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

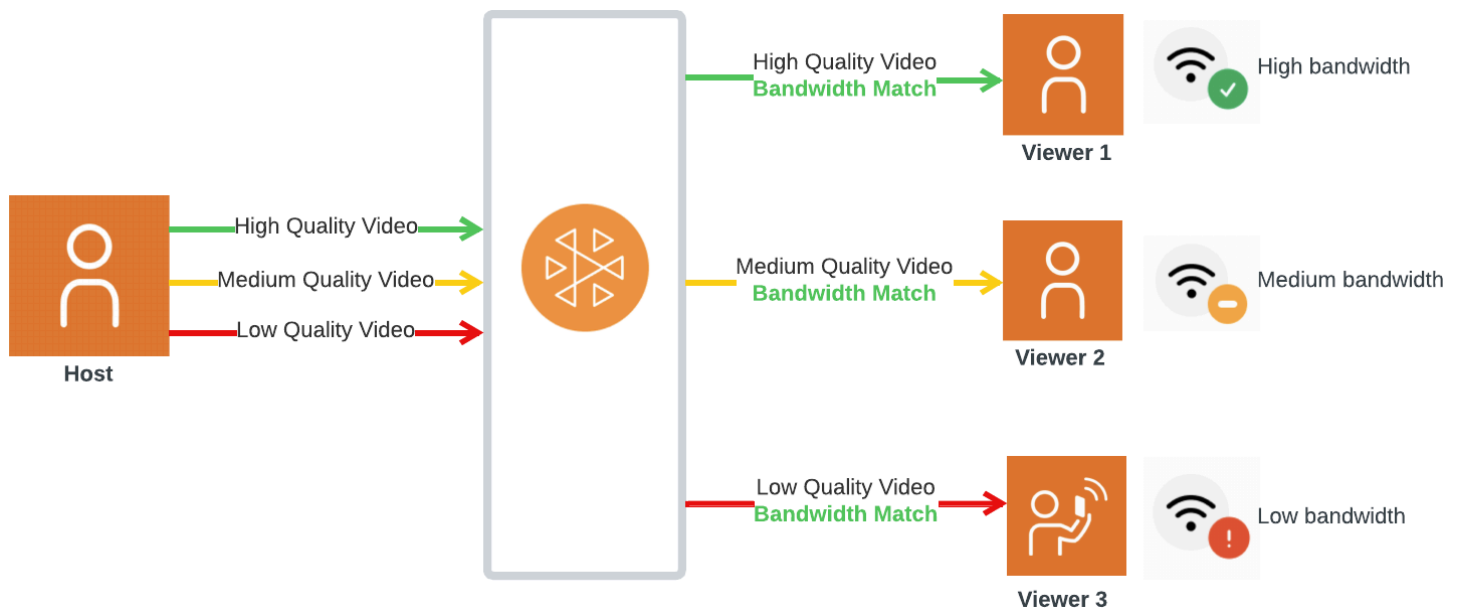
只有下列用戶端版本才支援此功能：

- iOS 和 Android 1.18.0+
- Web 1.12.0+

使用 IVS [即時廣播 SDK](#) 時，發布者可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 和 Edge 桌面瀏覽器 (適用於 Windows 和 macOS) 都支援 Simulcast 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內容 (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)
<p>高層級 (或自訂)：</p> <ul style="list-style-type: none"> • 最大位元速率：900,000 bps • 影格率：15 fps 	<p>高層級 (或自訂)：</p> <ul style="list-style-type: none"> • 最大位元速率：1,700,000 bps • 影格率：30 fps
<p>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)</p>	<p>中層：</p> <ul style="list-style-type: none"> • 最大位元速率：700,000 bps • 影格率：20 fps
<p>低層：</p> <ul style="list-style-type: none"> • 最大位元速率：100,000 bps • 影格率：15 fps 	<p>低層：</p> <ul style="list-style-type: none"> • 最大位元速率：200,000 bps • 影格率：15 fps

分層解析度

中層和低層的解析度會自動從高層縮減，以維持相同的長寬比。

如果中層和低層的解析度太接近高層，則會遭到排除。例如，如果設定的解析度為 320x180，則 SDK 也不會傳送較低的解析度層。

下表顯示不同解析度設定產生的分層解析度。列出的值適用於橫向螢幕，但可以反向套用至縱向螢幕。

輸入解析度	輸出層解析度：行動裝置	輸出層解析度：Web
720p (1280x720)	高 (1280x720)	高 (1280x720)
	低 (320x180)	中 (640x360)
		低 (320x180)
540p (960x540)	高 (960x540)	高 (960x540)
	低 (320x180)	低 (320x180)
360p (640x360)	高 (640x360)	高 (640x360)
	低 (360x180)	低 (360x180)
270p (480x270)	高 (480x270)	高 (480x270)
180p (320x180)	高 (320x180)	高 (320x180)

對於上面未映射的自訂輸入解析度，您可以[使用下列工具](#)進行計算。

設定 Simulcast 分層編碼

若要使用 Simulcast 分層編碼，您必須在用戶端上[啟用此功能](#)。如果啟用此功能，您會看到發布者的上傳頻寬用量增加，觀眾的影片凍結時間可能會減少。

Android

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

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

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

iOS

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

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

// Other Stage implementation code
```

Web

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

// Other Stage implementation code
```

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

```
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
```

變更訂閱用戶抖動緩衝區最低延遲

若要變更訂閱參與者的抖動緩衝區最低延遲，可以使用自訂 `subscribeConfiguration`。抖動緩衝區會決定播放開始之前要儲存的封包數量。最低延遲代表應儲存最少的資料量。變更最低延遲有助於在面臨封包遺失/連線問題時能更彈性地進行播放。

但是，增加抖動緩衝區大小也會增加播放開始之前的延遲。增加最低延遲可提供更高的彈性，代價是會影響影片播放的時間。請注意，在播放期間增加最低延遲會有類似的影響：播放會短暫暫停，以便填滿抖動緩衝區。

如果需要更多彈性，建議您先從 `MEDIUM` 的最低延遲預設值開始，並在播放開始之前設定訂閱組態。

請注意，只有在參與者為僅訂閱時，才會套用最低延遲。如果參與者自行發布，則不會套用最低延遲。這樣做是為了確保多個發布者可以彼此交談，而不會產生額外的延遲。

下列範例使用 MEDIUM 的最低延遲預設值。請參閱 SDK 參考文件，了解所有可能的值。

Web

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

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

Android

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

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

    return config;
}
```

iOS

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

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

    return config
}
```

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 。

IVS 資源與支援 | 即時串流

本文件列出資源，以協助支援您使用 Amazon IVS 即時串流功能。

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 建置引人注目的社交使用者產生的即時內容應用程式。此應用程式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 AWS](#) 中的連結可用來查詢與帳單或帳戶相關的非技術問題。對於技術問題，請使用上述開發論壇或支援連結。

IVS 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 <u>壓縮</u> 的音訊編碼標準。AAC 旨在作為 MP3 格式の後繼者，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 <u>位元速率</u> ，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 <u>壓縮</u> 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。您可以使用 CloudWatch 監控 IVS 指標；請參閱 監控 IVS 即時串流 和 監控 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	IVS 發布給 AmazonEventBridge 監控服務的自動通知。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 和 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 轉碼复用 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 物件標記法，一種開放式標準檔案格式，它使用人類可讀取的文字來傳輸資料物件，其中包含屬性值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、玻璃到玻璃延遲	資料傳輸中的延遲。IVS 將延遲範圍定義為： <ul style="list-style-type: none"> 低延遲：3 秒以下 即時延遲：300 毫秒以下 <p>顯示屏到顯示屏延遲是指從攝影機擷取即時串流到串流出現在檢視器螢幕上的延遲。</p>	✓	✓	
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，一種限制對 S3 儲存貯體 的存取的機制，以便只能透過 CloudFront CDN 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	<p>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。</p> <p>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 IVS 私有頻道中的建立或匯入 IVS 播放金鑰，並參閱 IVS 低延遲 API 參考中的播放金鑰對端點。</p>	✓		
播放 URL	<p>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。</p>	✓		
私有頻道	<p>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 IVS 私有頻道中的 IVS 私有頻道的工作流程。</p>	✓		
漸進式影片	<p>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。</p>	✓	✓	
配額	<p>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。</p>	✓	✓	✓

術語	描述	LL	RT	聊天
區域	<p>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。</p> <p>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。</p>	✓	✓	✓
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	<p>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。</p> <p>另請參閱用戶端合成。</p>		✓	
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，但可以設定更短的時間。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	<p>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。</p> <p>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。</p>	✓		
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。	✓	✓	
轉碼复用	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。「轉碼复用」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。轉碼复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	<p>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。</p> <p>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。</p>	✓		
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	<p>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。</p> <p>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。</p>	✓		
觀眾	從 IVS 接收 串流 的人員。	✓		
WebRTC	<p>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。它允許直接對等通訊，從而允許音訊和影片通訊在網頁內進行，而無需安裝外掛程式或下載原生應用程式。</p> <p>WebRTC 背後的技術是作為開放 Web 標準實作的，並且可以作為所有主要瀏覽器中的一般 JavaScript API 或作為原生用戶端 (如 Android 和 iOS) 的程式庫提供。</p>	✓	✓	

術語	描述	LL	RT	聊天
WHIP	<p>WebRTC-HTTP 擷取通訊協定，這是一種 HTTP 型通訊協定，可讓 WebRTC 型內容擷取進入串流服務和/或 CDN 中。WHIP 是為了標準化 WebRTC 擷取而開發的 IETF 草案。</p> <p>WHIP 可以與 OBS 等軟體相容，為桌面發布提供了一種替代方案 (IVS 廣播 SDK)。熟悉 OBS 的更資深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</p> <p>在不可或不偏好使用 IVS 廣播 SDK 的情形下，WHIP 也是有用的。例如，在涉及硬體編碼器的設定中，IVS 廣播 SDK 可能不適用。但是，如果編碼器支援 WHIP，您仍然可以直接從編碼器發布到 IVS。</p> <p>請參閱 IVS WHIP 支援 (即時串流)。</p>		✓	
WSS	<p>WebSocket Secure，一種用於透過加密的 TLS 連線建立 WebSocket 的協定。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。</p>			✓

IVS 文件歷史記錄 | 即時串流

下表說明了對 Amazon IVS 即時串流功能文件所進行的重大變更。我們會針對新版本和您傳送給我們的意見回饋，經常更新說明文件。

《即時串流使用者指南》變更

變更	描述	日期
廣播 SDK : Web 1.17.0	更新了 IVS 文件登陸頁面 上和低延遲串流廣播 SDK : Web 指南中的版本編號和成品連結。也請參閱 版本備註 。	2024 年 10 月 10 日
廣播 SDK : Android 1.23.0、iOS 1.23.0	更新了 IVS 文件登陸頁面 上和低延遲串流廣播 SDK : Android 指南和 iOS 指南中的版本編號和成品連結。也請參閱 版本備註 。 對於 Android，我們新增了 使用包含偵錯符號的 SDK 章節。	2024 年 10 月 10 日
Service Quotas	新增了「參與者發布位元速率」的配額。	2024 年 9 月 25 日
監控 IVS 即時串流	新增了 PublishFramerate CloudWatch 指標。	2024 年 9 月 13 日
廣播 SDK : Android 1.22.0、iOS 1.22.0	更新了 IVS 文件登陸頁面 上和低延遲串流廣播 SDK : Android 指南和 iOS 指南中的版本編號和成品連結。也請參閱 版本備註 。	2024 年 9 月 11 日

我們也更新了廣播 SDK：
Android 指南中的「開始使用 >
安裝程式庫」一節。

[廣播 SDK : Web 1.16.0](#)

更新了 [IVS 文件登陸頁面](#)上和
低延遲串流廣播 SDK：[Web](#) 指
南中的版本編號和成品連結。
也請參閱[版本備註](#)。

2024 年 9 月 11 日

[RTMP 擷取](#)

我們新增了 [IVS 串流擷取](#)頁
面。在此頁面下有兩個頁面，
即 RTMP (新) 和 WHIP。

2024 年 9 月 9 日

我們在[搭配 IVS 即時串流使用
EventBridge](#) 章節中，新增了
「IVS 舞台更新」事件，「參
與者發布錯誤」。

在 [Service Quotas](#) 中，為五
個新的 API 操作和 1 個新的
IngestConfiguration 配額 (在
「其他配額」中) 新增了 TPS
值。

API 變更的說明位於 [API 參
考](#)資料表中。

[即時串流最佳化](#)

更新了多處與 Simulcast 相關
的內容，並新增了「分層解析
度」內容。

2024 年 8 月 22 日

[主控台內發布/訂閱](#)

我們在開始使用 IVS 即時串
流章節中，新增了主控台內發
布和訂閱功能以[發布和訂閱影
片](#)相關的內容。

2024 年 8 月 19 日

[廣播 SDK : Web 1.15.0](#)

更新了 [IVS 文件登陸頁面](#)上和低延遲串流廣播 SDK : [Web](#) 指南中的版本編號和成品連結。也請參閱[版本備註](#)。

2024 年 8 月 15 日

我們也在廣播 SDK : Web 指南章節中新增了[參與者訂閱組態](#)章節。

我們在串流最佳化章節中新增了[變更訂閱用戶抖動緩衝區最低延遲](#)章節。這包括 Web、Android 和 iOS 廣播 SDK 的相關資訊。

[廣播 SDK : Android 1.21.0、iOS 1.21.0](#)

更新了 [IVS 文件登陸頁面](#)上和低延遲串流廣播 SDK : [Android](#) 指南和 [iOS](#) 指南中的版本編號和成品連結。也請參閱[版本備註](#)。

2024 年 8 月 15 日

我們也在廣播 SDK : [Android](#) 指南和 [iOS](#) 指南中新增了「參與者訂閱組態」新章節。

[錄製說明](#)

在個別參與者錄製 (在 [1 : 建立 S3 儲存貯體](#)中) 和複合錄製 (在[先決條件](#)中，步驟 3) 章節中新增了有關使用現有 S3 儲存貯體的備註。物件擁有權設定必須是強制執行的儲存貯體擁有者，或是偏好的儲存貯體擁有者。

2024 年 8 月 13 日

[廣播 SDK : Web 1.14.0](#)

更新了 [IVS 文件登陸頁面](#)上和低延遲串流廣播 SDK : [Web](#) 指南中的版本編號和成品連結。也請參閱[版本備註](#)。

2024 年 7 月 18 日

廣播 SDK : Android 1.20.0、iOS 1.20.0	更新了 IVS 文件登陸頁面 上和低延遲串流廣播 SDK : Android 指南和 iOS 指南中的版本編號和成品連結。也請參閱 版本備註 。	2024 年 7 月 18 日
開始使用即時串流	在「權杖結構描述：承載」和「使用 IVS 即時串流 API 建立權杖」中，新增了「分發參與者權杖」中屬性的相關資訊。	2024 年 7 月 12 日
Service Quotas	舞台訂閱用戶數上限從 10,000 增加到 25,000。	2024 年 6 月 27 日
使用金鑰對產生參與者權杖	我們在開始使用 IVS 即時串流章節中，更新了 分發參與者權杖 ，以解釋權杖的兩種產生方式 (API 和金鑰對)，並新增了「使用金鑰對建立權杖」章節。	2024 年 6 月 26 日
個別參與者錄製	新增了與 錄製 相關的文件章節，其中包含 個別參與者錄製 (新) 和複合錄製 (既有) 的子文件。我們也在 搭配 IVS 使用 EventBridge 章節中新增了「參與者錄製狀態變更」事件和範例。 API 變更的說明位於 API 參考 資料表中。	2024 年 6 月 20 日
Service Quotas	舞台配額從 100 增加到 1,000。	2024 年 6 月 14 日

[廣播 SDK : Android 1.19.0、iOS 1.19.0](#)

更新了 [IVS 文件登陸頁面](#)上和低延遲串流廣播 SDK : [Android](#) 指南和 [iOS](#) 指南中的版本編號和成品連結。也請參閱[版本備註](#)。

2024 年 6 月 13 日

[廣播 SDK : Web 1.13.0](#)

更新了 [IVS 文件登陸頁面](#)上和低延遲串流廣播 SDK : [Web](#) 指南中的版本編號和成品連結。也請參閱[版本備註](#)。

2024 年 6 月 13 日

我們在這份指南中，更新了[錯誤處理](#)中與新的 ERROR 舞台事件相關的資訊。

[廣播 SDK : Web 1.12.0](#)

更新了 [IVS 文件登陸頁面](#)上和低延遲串流廣播 SDK : [Web](#) 指南中的版本編號和成品連結。也請參閱[版本備註](#)。

2024 年 5 月 20 日

我們在這份指南中，更新了[處理網路問題](#)中與舞台連線 ERRORED 狀態相關的資訊。

[即時串流最佳化](#)

將[預設層、品質和影格速率](#)章節中行動裝置 (低層) 的最大位元速率從 150,000 變更為 100,000 bps。

2024 年 5 月 16 日

[廣播 SDK : Android 1.18.0、iOS 1.18.0](#)

更新了 [IVS 文件登陸頁面](#)上和低延遲串流廣播 SDK : [Android](#) 指南和 [iOS](#) 指南中的版本編號和成品連結。也請參閱[版本備註](#)。

2024 年 5 月 16 日

廣播 SDK : Web 1.11.0	更新了 IVS 文件登陸頁面 上和即時串流廣播 SDK : Web 指南中的版本編號和成品連結。也請參閱 版本備註 。	2024 年 5 月 6 日
廣播 SDK : Web 1.10.1	更新了 IVS 文件登陸頁面 上和即時串流廣播 SDK : Web 指南中的版本編號和成品連結。也請參閱 版本備註 。	2024 年 4 月 30 日
廣播 SDK : Android 1.15.2、iOS 1.15.2	更新了 IVS 文件登陸頁面 上和低延遲串流廣播 SDK : Android 指南和 iOS 指南中的版本編號和成品連結。也請參閱 版本備註 。	2024 年 4 月 30 日
廣播 SDK : iOS 指南	我們更新了 發布媒體串流 章節中的程式碼範例。	2024 年 4 月 26 日
廣播 SDK : Android 1.17.0、iOS 1.17.0	在 Android 版和 iOS 版即時串流廣播 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 。	2024 年 4 月 22 日
伺服器端合成	對 SSC 進行了各種變更 (特別是在「版面配置」章節中)，以解釋 PiP 和網格版面配置。 在廣播 SDK : Web 指南中，新增了 伺服器端轉譯支援 章節。	2024 年 3 月 26 日
OBS 和 WHIP 支援	新增了備註，說明 OBS 中可能發生的與 WHIP 相關的品質問題 (例如間歇性影片凍結)。	2024 年 3 月 22 日

[廣播 SDK : Android](#)[1.16.0、iOS 1.16.0、Web 1.10.0](#)

更新了即時串流廣播 SDK : [Android](#) 指南、[iOS](#) 指南、[Web](#) 指南中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](#)。

2024 年 3 月 21 日

[廣播 SDK : Android](#)[1.15.1、iOS 1.15.1](#)

在 [Android](#) 版和 [iOS](#) 版即時串流廣播 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](#)。

2024 年 3 月 13 日

[廣播 SDK : 行動音訊模式](#)

在「音訊模式預設」章節中，新增了音量鍵預設類別的相關資訊，以及 iOS 中已知與視訊聊天預設相關的問題。在「進階使用案例」章節中，新增了避免不正確組態的備註，並新增了與「iOS 回音消除」和「iOS 自訂音訊來源」相關的章節。

2024 年 3 月 1 日

[廣播 SDK : Android](#)[1.15.0、iOS 1.15.0、Web 1.9.0](#)

更新了即時串流廣播 SDK : [Android](#) 指南、[iOS](#) 指南、[Web](#) 指南中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](#)。

2024 年 2 月 22 日

[OBS 和 WHIP 支援](#)

新增了一個頁面。本文件說明如何使用 OBS 等 WHIP 相容編碼器，來發布至 IVS 即時串流。WHIP (WebRTC-HTTP 擷取通訊協定) 是為了標準化 WebRTC 擷取而開發的 IETF 草案。

2024 年 2 月 6 日

[廣播 SDK : Android 1.14.1、iOS 1.14.1、Web 1.8.0](#)

更新了即時串流廣播 SDK : [Android](#) 指南、[iOS](#) 指南、[Web](#) 指南中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](#)。

2024 年 2 月 1 日

對於 Android 指南，我們新增了已知問題 (影片大小小於 176x176)。

對於 Web 指南，我們新增了已知問題。因應措施是在叫用 `getUserMedia` 或 `getDisplayMedia` 時，將影片解析度限制為 720p。

我們在即時串流最佳化章節中，更新了 [設定 Simulcast 分層編碼](#) 內容 (此功能目前預設為停用)。

[廣播 SDK : Android](#)[1.13.4、iOS 1.13.4、Web 1.7.0](#)

更新了即時串流廣播 SDK : [Android](#) 指南、[iOS](#) 指南、[Web](#) 指南中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](#)。

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

[階段運作狀態：新的 CloudWatch 指標](#)

將 PacketLoss (Stage) 指標重新命名為 DownloadPacketLoss (Stage)，並發布了 IVS 即時串流的其他 CloudWatch 指標：

2023 年 12 月 7 日

- DownloadPacketLoss (Stage,Participant)
- DroppedFrames (Stage,Participant)
- SubscribeBitrate (Stage,Participant,MediaType)

請參閱 [監控 IVS 即時串流](#)。

[IAM 受管政策](#)

新增兩個受管政策，即 IVSReadOnlyAccess 和 IVSFullAccess。請參閱：

2023 年 12 月 5 日

- 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](#)。

[廣播 SDK : Android 1.13.2 和 iOS 1.13.2](#)

在 [Android](#) 版和 [iOS](#) 版即時串流廣播 SDK 指南中，更新了新版本的版本編號和成品連結。

2023 年 12 月 4 日

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](#)。

[廣播 SDK : Android 1.13.1](#)

在 [Android](#) 版即時串流廣播 SDK 指南中，更新了新版本的版本編號和成品連結。

2023 年 11 月 21 日

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](#)。

[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 [Android](#) 版和 [iOS](#) 版即時串流廣播 SDK 指南中，更新了新版本的版本編號和成品連結。

2023 年 11 月 17 日

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。

[複合錄製](#)

進行下列變更：

2023 年 11 月 16 日

- 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](#)。

[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：

2023 年 11 月 16 日

- [開始使用](#)：我們已將 SSC 端點新增至「設定 IAM 許可」中的政策。
- [搭配 IVS 使用 Amazon EventBridge](#)：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](#)：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。

[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)」。

2023 年 11 月 9 日

在[廣播 Web 指南](#)中，我們新增了「行動 Web 限制」。

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 指標：IVS 即時串流 ，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	<p>在下列即時串流廣播 SDK 指南中，更新了新版本的版本編號和成品連結：Web。</p> <p>Amazon IVS 文件登陸頁面會指向最新版本的廣播 SDK 參考。</p> <p>另請參閱此版本的 Amazon IVS 版本備註。</p> <p>在網路指南的「從裝置擷取 MediaStream」中，我們也刪除了這兩max行；最佳實踐是僅指定。ideal</p> <p>在「即時串流優化」中，我們新增了一個部分：優化音訊位元速率及立體聲支援。</p>	2023 年 10 月 16 日
階段運作狀態：新的 CloudWatch 指標	推出 IVS 即時串流的 CloudWatch 指標。請參閱 監控 IVS 即時串流 。	2023 年 10 月 12 日

[廣播 SDK : Android 1.12.1](#)

在 [Android](#) 版即時串流廣播 SDK 指南中，更新了新版本的版本編號和成品連結。也新增了 [使用藍牙麥克風](#) 的新章節。

2023 年 10 月 12 日

[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](#)。

[廣播 SDK : Web 1.5.2](#)

在下列即時串流廣播 SDK 指南中，更新了新版本的版本編號和成品連結：[Web](#)。

2023 年 9 月 14 日

[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](#)。

[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](#)

在下列即時串流廣播 SDK 指南中，更新了新版本的版本編號和成品連結：[Web](#)、[Android](#)，以及 [iOS](#)。

2023 年 8 月 23 日

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](#)。

[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。

2023 年 8 月 7 日

如需其他文件變更，請參閱 [文件歷史記錄 \(低延遲串流\)](#)。

[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。

2023 年 8 月 7 日

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](#)。

《IVS 即時串流 API 參考》變更

API 變更	說明	日期
更新事件和影片物件	<p>我們在「事件」物件中為 <code>errorCode</code> 新增了更多有效值。</p> <p>我們在「影片」物件中說明了 <code>height</code> 和 <code>width</code> 必須是偶數。</p>	2024 年 10 月 2 日
RTMP 擷取	<p>我們新增了兩個物件：<code>IngestConfiguration</code> 和 <code>IngestConfigurationSummary</code>。我們新增了五個 <code>IngestConfiguration</code> 端點 (建立、刪除、取得、列出和更新)。</p> <p>我們更新了 <code>DeleteStage</code> (操作說明) 和 <code>DisconnectParticipant</code> (操作說明和 <code>participantId</code>)。</p> <p>我們修改了「參與者」物件 (新增了 <code>protocol</code> 欄位)；這會影響 <code>GetParticipant</code> 回應。</p> <p>我們修改了 <code>StageEndpoints</code> 物件 (新增了 <code>rtmp</code> 和 <code>rtmps</code> 欄位)；這會影響 <code>CreateStage</code>、<code>GetStage</code> 和 <code>UpdateStage</code> 回應。我們也更新了此物件的描述 (新增了快取建議)。</p>	2024 年 9 月 9 日
使用金鑰對產生參與者權杖	<p>我們新增了三個物件 (<code>PublicKey</code>、<code>PublicKeySummary</code>、<code>StageEndpoints</code>) 和四個端點：<code>DeletePublicKey</code>、<code>GetPublicKey</code>、<code>ImportPublicKey</code>、<code>ListPublicKeys</code>。我們修改了「舞台」物件 (新增了 <code>endpoints</code> 欄位)；這會影響 <code>CreateStage</code>、<code>GetStage</code> 和 <code>UpdateStage</code> 回應。</p>	2024 年 6 月 26 日
個別參與者錄製	<p>我們新增了一個物件 (<code>AutoParticipantRecordingConfiguration</code>)，並且修改了三個物件 (<code>Participant</code>、<code>ParticipantSummary</code>、<code>Stage</code>)。這會影響五個端點：<code>CreateStage</code> 請求和回應、<code>GetParticipant</code> 回應、<code>GetStage</code> 回</p>	2024 年 6 月 20 日

API 變更	說明	日期
	應、ListParticipants 請求和回應，以及 UpdateStage 請求和回應。	
將 svs 從 ARN 模式中移除	更新了指定 [is]vs 的 ARN 模式以指定 ivs。這會影響所有三個標籤端點和 ChannelDestination Configuration\$channelArn 欄位。	2024 年 4 月 25 日
伺服器端合成更新	我們新增了一個物件：PipConfiguration。 我們修改了兩個物件 (LayoutConfiguration、GridConfiguration)。這會影響 GetComposition 的回應，以及 StartComposition 的請求和回應。	2024 年 3 月 13 日
複合錄製	我們新增了 4 個 StorageConfiguration 端點和 7 個物件 (DestinationDetail、RecordingConfiguration、S3DestinationConfiguration、S3Detail、S3StorageConfiguration、StorageConfiguration、StorageConfigurationSummary)。 我們修改了 3 個物件 (Composition、Destination、DestinationConfiguration)。這會影響 GetComposition 的回應，以及 StartComposition 的請求和回應。	2023 年 11 月 16 日
伺服器端合成	我們新增了 8 個 Composition 和 EncoderConfiguration 端點以及 11 個物件 (ChannelDestinationConfiguration、Composition、CompositionSummary、Destination、DestinationConfiguration、DestinationSummary、EncoderConfiguration、EncoderConfigurationSummary、GridConfiguration、LayoutConfiguration 和 Video)。	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 GetParticipant 回應。	2023 年 10 月 12 日

API 變更	說明	日期
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	<p>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。</p> <p>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。</p>	2023 年 8 月 7 日

IVS 版本備註 | 即時串流

本文件包含所有 Amazon IVS 即時串流功能版本備註，以最新版本為先，依發行日期整理。

2024 年 10 月 10 日

IVS 廣播 SDK : Web 1.17.0 (即時串流)

平台	下載與變更
Web 廣播 SDK 1.17.0	<p>參考文件：https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference</p> <ul style="list-style-type: none">次要錯誤修正。

2024 年 10 月 10 日

Amazon IVS 廣播 SDK : Android 1.23.0、iOS 1.23.0 (即時串流)

平台	下載與變更
Android 廣播 SDK 1.23.0	<p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.23.0/android/</p> <ul style="list-style-type: none">我們在此版本中，也開始發布包含偵錯符號的 Android 廣播 SDK 版本。請參閱使用包含偵錯符號的 SDK。次要錯誤修正。
iOS 廣播 SDK 1.23.0	<p>適用於即時串流的下載：https://broadcast.live-video.net/1.23.0/AmazonIVSBroadcast-Stages.xcframework.zip</p> <p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.23.0/ios/</p>

平台	下載與變更
	<ul style="list-style-type: none"> 次要錯誤修正。

廣播 SDK 大小：Android

架構	壓縮大小	未壓縮大小
arm64-v8a	5.432 MB	13.560 MB
armeabi-v7a	4.707 MB	9.451 MB
x86_64	5.626 MB	14.459 MB
x86	5.838 MB	14.908 MB

廣播 SDK 大小：iOS

架構	壓縮大小	未壓縮大小
arm64	3.542 MB	8.316 MB

2024 年 9 月 11 日

Amazon IVS 廣播 SDK：Android 1.22.0、iOS 1.22.0 (即時串流)

平台	下載與變更
Android 廣播 SDK 1.22.0	<p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.22.0/android/</p> <ul style="list-style-type: none"> 修正了以下錯誤：特定 Android 裝置在切換攝影機輸入後，在預覽中顯示黑色影格。 次要錯誤修正。

平台	下載與變更
iOS 廣播 SDK 1.22.0	<p>適用於即時串流的下載：https://broadcast.live-video.net/1.22.0/AmazonIVSBroadcast-Stages.xcframework.zip</p> <p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.22.0/ios/</p> <ul style="list-style-type: none"> 次要錯誤修正。

廣播 SDK 大小：Android

架構	壓縮大小	未壓縮大小
arm64-v8a	5.359 MB	13.392 MB
armeabi-v7a	4.636 MB	9.325 MB
x86_64	5.548 MB	14.268 MB
x86	5.754 MB	14.710 MB

廣播 SDK 大小：iOS

架構	壓縮大小	未壓縮大小
arm64	3.488 MB	8.199 MB

2024 年 9 月 11 日

IVS 廣播 SDK : Web 1.16.0 (即時串流)

平台	下載與變更
Web 廣播 SDK 1.16.0	參考文件： https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference <ul style="list-style-type: none"> 次要錯誤修正。

2024 年 9 月 9 日

RTMP 擷取

除了使用 IVS 廣播 SDK 之外，您現在也可以從 RTMP 來源 (也包括原本支援的 WHIP)，將影片發布至 IVS 舞台。如需文件變更，請參閱[文件歷史記錄](#) (使用者指南和 API 參考資料表)。

2024 年 8 月 19 日

主控台內發布/訂閱

您現在可以透過 IVS 主控台進行發布和訂閱。在開始使用 IVS 即時串流中，請參閱[發布和訂閱影片](#)。

2024 年 8 月 15 日

IVS 廣播 SDK : Web 1.15.0 (即時串流)

平台	下載與變更
Web 廣播 SDK 1.15.0	參考文件： https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference <ul style="list-style-type: none"> 修正了一個競爭條件，在重複呼叫 <code>join()</code> 時，此條件會影響發布者媒體品質。連續呼叫 <code>join()</code> 不再會再次觸發 <code>STAGE_PAR</code>

平台	下載與變更
	<p>TICIPANT_JOINED 事件，以及伴隨的發布和串流狀態變更。</p> <ul style="list-style-type: none"> 修正了以下錯誤：在權杖 attributes 欄位中使用非文字字元時，導致剖析參與者權杖時出現問題。 新增了參與者訂閱用戶的設定方法。最初，只能設定抖動緩衝區最低延遲。請參閱 SDK 參考文件廣播 SDK：Web 指南中的參與者訂閱組態，以及串流最佳化章節中的變更訂閱用戶抖動緩衝區最低延遲。

2024 年 8 月 15 日

Amazon IVS 廣播 SDK：Android 1.21.0、iOS 1.21.0 (即時串流)

平台	下載與變更
<p>Android 廣播 SDK 1.21.0</p>	<p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.21.0/android/</p> <ul style="list-style-type: none"> 修正以下影響配備 MT6765 晶片組裝置的錯誤：訂閱用戶預覽在某些情況下會變成黑色影格。 新增了參與者訂閱用戶的設定方法。最初，只能設定抖動緩衝區最低延遲。請參閱 SDK 參考文件廣播 SDK：Android 指南中的參與者訂閱組態，以及串流最佳化章節中的變更訂閱用戶抖動緩衝區最低延遲。 次要錯誤修正。
<p>iOS 廣播 SDK 1.21.0</p>	<p>適用於即時串流的下載：https://broadcast.live-video.net/1.21.0/AmazonIVSBroadcast-Stages.xcframework.zip</p>

平台	下載與變更
	<p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.21.0/ios/</p> <ul style="list-style-type: none"> • 新增了參與者訂閱用戶的設定方法。最初，只能設定抖動緩衝區最低延遲。請參閱 SDK 參考文件廣播 SDK：iOS 指南中的參與者訂閱組態，以及串流最佳化章節中的變更訂閱用戶抖動緩衝區最低延遲。 • 次要錯誤修正。

廣播 SDK 大小：Android

架構	壓縮大小	未壓縮大小
arm64-v8a	5.350 MB	13.378 MB
armeabi-v7a	4.628 MB	9.312 MB
x86_64	5.538 MB	14.253 MB
x86	5.744 MB	14.694 MB

廣播 SDK 大小：iOS

架構	壓縮大小	未壓縮大小
arm64	3.485 MB	8.199 MB

2024 年 7 月 18 日

IVS 廣播 SDK : Web 1.14.0 (即時串流)

平台	下載與變更
Web 廣播 SDK 1.14.0	<p>參考文件：https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference</p> <ul style="list-style-type: none">• API 文件改善。• 修正了以下問題：連線重設期間報告影片和音訊統計資料異常值。• 次要相依性更新。

2024 年 7 月 18 日

Amazon IVS 廣播 SDK : Android 1.20.0、iOS 1.20.0 (即時串流)

平台	下載與變更
Android 廣播 SDK 1.20.0	<p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.20.0/android</p> <ul style="list-style-type: none">• 修正了以下錯誤：阻止廣播 SDK 在採用 Intel 處理器的 Chromebook 上執行。• 次要錯誤修正。
iOS 廣播 SDK 1.20.0	<p>適用於即時串流的下載：https://broadcast.live-video.net/1.20.0/AmazonIVSBroadcast-Stages.xcframework.zip</p> <p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.20.0/ios</p> <ul style="list-style-type: none">• 次要錯誤修正。

廣播 SDK 大小：Android

架構	壓縮大小	未壓縮大小
arm64-v8a	5.318 MB	13.299 MB
armeabi-v7a	4.605 MB	9.254 MB
x86_64	5.507 MB	14.168 MB
x86	5.715 MB	14.608 MB

廣播 SDK 大小：iOS

架構	壓縮大小	未壓縮大小
arm64	3.465 MB	8.164 MB

2024 年 6 月 26 日

使用金鑰對產生參與者權杖

您現在可以使用金鑰對，在自己的伺服器應用程式上產生參與者權杖。這可讓您避免每次需要參與者權杖時就要呼叫 `CreateParticipantToken`。如需文件變更，請參閱[文件歷史記錄](#) (使用者指南和 API 參考資料表)。

2024 年 6 月 20 日

個別參與者錄製

個別參與者錄製允許 IVS 即時串流客戶，將 IVS 舞台發布者個別錄製到 S3 儲存貯體。請參閱《[即時串流 API 參考](#)》中的[錄製](#)、[個別參與者錄製](#)和變更。(如需具體的文件變更，請參閱[文件歷史記錄](#)。)

2024 年 6 月 13 日

Amazon IVS 廣播 SDK : Android 1.19.0、iOS 1.19.0 (即時串流)

平台	下載與變更
Android 廣播 SDK 1.19.0	<p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.19.0/android</p> <ul style="list-style-type: none"> • 最新的 Android 版本要求在擷取畫面時顯示的通知中加入一個圖示。您現在可以視需要在 <code>Session # createServiceNotificationBuilder</code> 傳回的 <code>Notification.Builder</code> 上呼叫 <code>setSmallIcon</code>，來自訂圖示。 • 縮短了從 WiFi 轉換為行動連線之裝置的連線復原時間。此變更需要 <code>CHANGE_NETWORK_STATE</code> 許可。
iOS 廣播 SDK 1.19.0	<p>適用於即時串流的下載：https://broadcast.live-video.net/1.19.0/AmazonIVSBroadcast-Stages.xcframework.zip</p> <p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.19.0/ios</p> <ul style="list-style-type: none"> • 次要錯誤修正。

廣播 SDK 大小 : Android

架構	壓縮大小	未壓縮大小
arm64-v8a	5.304 MB	13.340 MB
armeabi-v7a	4.598 MB	9.299 MB
x86_64	5.495 MB	14.207 MB

架構	壓縮大小	未壓縮大小
x86	5.694 MB	14.625 MB

廣播 SDK 大小：iOS

架構	壓縮大小	未壓縮大小
arm64	3.393 MB	7.949 MB

2024 年 6 月 13 日

IVS 廣播 SDK：Web 1.13.0 (即時串流)

平台	下載與變更
Web 廣播 SDK 1.13.0	<p>參考文件：https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference</p> <ul style="list-style-type: none"> 更新了 StageEvents.STAGE_PARTICIPANT_SUBSCRIBE_STATE_CHANGED 和 StageEvents.STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED 的事件變更行為持續時間。參與者現在可以在 ATTEMPTING_SUBSCRIBE 或 ATTEMPTING_PUBLISH 狀態保持更長時間，直到 ERRORED 事件觸發為止。 新增了 StageEvents.ERROR 事件，用於接聽 SDK 所遭遇的錯誤。如需詳細資訊，請參閱即時廣播 SDK：Web 指南中的錯誤處理。

2024 年 5 月 20 日

IVS 廣播 SDK : Web 1.12.0 (即時串流)

平台	下載與變更
Web 廣播 SDK 1.12.0	<p>參考文件：https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference</p> <ul style="list-style-type: none">• 改善了發布和訂閱操作的重試處理。• 改善了分析，特別是延遲和音訊品質測量。

2024 年 5 月 16 日

Amazon IVS 廣播 SDK : Android 1.18.0、iOS 1.18.0 (即時串流)

平台	下載與變更
Android 廣播 SDK 1.18.0	<p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.18.0/android</p> <ul style="list-style-type: none">• 如果連線的舞台被 AWS 控制平面刪除，或使用中的權杖被撤銷，SDK 現在會傳送特定錯誤代碼。• 次要錯誤修正。
iOS 廣播 SDK 1.18.0	<p>適用於即時串流的下載：https://broadcast.live-video.net/1.18.0/AmazonIVSBroadcast-Stages.xcframework.zip</p> <p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.18.0/ios</p> <ul style="list-style-type: none">• 如果連線的舞台被 AWS 控制平面刪除，或使用中的權杖被撤銷，SDK 現在會傳送特定錯誤代碼。

平台	下載與變更
	<ul style="list-style-type: none"> 新增了 IVSCamera setVideoZoomFactor 方法和相關聯的 IVSCameraDelegate 方法。

廣播 SDK 大小：Android

架構	壓縮大小	未壓縮大小
arm64-v8a	5.275 MB	13.279 MB
armeabi-v7a	4.573 MB	9.254 MB
x86_64	5.472 MB	14.142 MB
x86	5.664 MB	14.554 MB

廣播 SDK 大小：iOS

架構	壓縮大小	未壓縮大小
arm64	3.393 MB	7.916 MB

2024 年 5 月 6 日

IVS 廣播 SDK：Web 1.11.0 (即時串流)

平台	下載與變更
Web 廣播 SDK 1.11.0	<p>參考文件：https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference</p> <ul style="list-style-type: none"> 修正了 SDK 未嘗試在舞台 DISCONNECT 上復原的邊緣案例。

平台	下載與變更
	<ul style="list-style-type: none"> 更新了 <code>join()</code> 逾時錯誤的錯誤訊息。SDK 現在會傳回「操作逾時」，而不是「10 秒後 <code>InitialConnectTimedOut</code>」。

2024 年 4 月 30 日

IVS 廣播 SDK : Web 1.10.1 (即時串流)

平台	下載與變更
Web 廣播 SDK 1.10.1	參考文件： https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference <ul style="list-style-type: none"> 次要錯誤修正。

2024 年 4 月 30 日

Amazon IVS 廣播 SDK : Android 1.15.2、iOS 1.15.2 (即時串流)

平台	下載與變更
Android 廣播 SDK 1.15.2	參考文件： https://aws.github.io/amazon-ivs-broadcast-docs/1.15.2/android <ul style="list-style-type: none"> 次要錯誤修正。除非您有特定原因必須升級至此版本，否則請使用發行的最高版本。
iOS 廣播 SDK 1.15.2	適用於即時串流的下載： https://broadcast.live-video.net/1.15.2/AmazonIVSBroadcast-Stages.xcframework.zip 參考文件： https://aws.github.io/amazon-ivs-broadcast-docs/1.15.2/ios

平台	下載與變更
	<ul style="list-style-type: none"> 次要錯誤修正。除非您有特定原因必須升級至此版本，否則請使用發行的最高版本。

廣播 SDK 大小：Android

架構	壓縮大小	未壓縮大小
arm64-v8a	5.244 MB	13.198 MB
armeabi-v7a	4.543 MB	9.192 MB
x86_64	5.437 MB	14.051 MB
x86	5.631 MB	14.461 MB

廣播 SDK 大小：iOS

架構	壓縮大小	未壓縮大小
arm64	3.359 MB	7.836 MB

2024 年 4 月 22 日

Amazon IVS 廣播 SDK：Android 1.17.0、iOS 1.17.0 (即時串流)

平台	下載與變更
Android 廣播 SDK 1.17.0	<p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.17.0/android</p> <ul style="list-style-type: none"> 修正了以下錯誤：發布時發生罕見當機情況。

平台	下載與變更
iOS 廣播 SDK 1.17.0	<p>適用於即時串流的下載：https://broadcast.live-video.net/1.17.0/AmazonIVSBroadcast-Stages.xcframework.zip</p> <p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.17.0/ios</p> <ul style="list-style-type: none"> 此 AmazonIVSBroadcast 架構現在包含 Apple 要求的隱私權資訊清單。

廣播 SDK 大小：Android

架構	壓縮大小	未壓縮大小
arm64-v8a	5.273 MB	13.275 MB
armeabi-v7a	4.571 MB	9.251 MB
x86_64	5.468 MB	14.137 MB
x86	5.662 MB	14.549 MB

廣播 SDK 大小：iOS

架構	壓縮大小	未壓縮大小
arm64	3.388 MB	7.916 MB

2024 年 3 月 21 日

Amazon IVS 廣播 SDK : Android 1.16.0、iOS 1.16.0、Web 1.10.0 (即時串流)

平台	下載與變更
Web 廣播 SDK 1.10.0	<p>參考文件：https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference</p> <ul style="list-style-type: none"> 修正了在取消訂閱或離開舞台後清除連線時出現的間歇性錯誤。
Android 廣播 SDK 1.16.0	<p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.16.0/android</p> <ul style="list-style-type: none"> 修正了以下問題：執行 Android 14、採用 Exynos 變體的 Samsung 裝置上出現預覽凍結。 新增了查詢攝影機縮放功能和設定縮放係數的功能。
iOS 廣播 SDK 1.16.0	<p>適用於即時串流的下載：https://broadcast.live-video.net/1.16.0/AmazonIVSBroadcast-Stages.xcframework.zip</p> <p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.16.0/ios</p> <ul style="list-style-type: none"> 次要錯誤修正。

廣播 SDK 大小：Android

架構	壓縮大小	未壓縮大小
arm64-v8a	5.253 MB	13.21 MB

架構	壓縮大小	未壓縮大小
armeabi-v7a	4.551 MB	9.204 MB
x86_64	5.447 MB	14.070 MB
x86	5.640 MB	14.480 MB

廣播 SDK 大小：iOS

架構	壓縮大小	未壓縮大小
arm64	3.361 MB	7.836 MB

2024 年 3 月 13 日

Amazon IVS 廣播 SDK：Android 1.15.1、iOS 1.15.1 (即時串流)

平台	下載與變更
Android 廣播 SDK 1.15.1	<p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.15.1/android</p> <ul style="list-style-type: none"> 修正了以下錯誤：訂閱遠端參與者時發生罕見當機情況。
iOS 廣播 SDK 1.15.1	<p>適用於即時串流的下載：https://broadcast.live-video.net/1.15.1/AmazonIVSBroadcast-Stages.xcframework.zip</p> <p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.15.1/ios</p> <ul style="list-style-type: none"> 修正了以下錯誤：訂閱遠端參與者時發生罕見當機情況。

廣播 SDK 大小：Android

架構	壓縮大小	未壓縮大小
arm64-v8a	5.243 MB	13.194 MB
armeabi-v7a	4.541 MB	9.188 MB
x86_64	5.628 MB	14.455 MB
x86	5.434 MB	14.046 MB

廣播 SDK 大小：iOS

架構	壓縮大小	未壓縮大小
arm64	3.358 MB	7.820 MB

2024 年 3 月 13 日

伺服器端合成 API 更新

我們將新的屬性引入了 GridConfiguration，同時引入了新的子母畫面版面配置，增強了合成的自訂選項。如需具體文件變更，請參閱[文件歷史記錄](#) (請參閱 API 參考變更資料表)。

重要事項：確保應用程式不依賴目前版面配置的特定功能，例如動態磚的大小和位置。您可以隨時對版面配置進行視覺化改善。

2024 年 3 月 8 日

伺服器端合成版面配置更新

今天，我們啟用了對預設網格版面配置的變更，詳細資訊見 [2024 年 2 月 7 日](#) 條目所述。

2024 年 2 月 22 日

Amazon IVS 廣播 SDK : Android 1.15.0、iOS 1.15.0、Web 1.9.0 (即時串流)

平台	下載與變更
Web 廣播 SDK 1.9.0	<p>參考文件：https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference</p> <ul style="list-style-type: none">• 改善了內部錯誤處理。
Android 廣播 SDK 1.15.0	<p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.15.0/android</p> <ul style="list-style-type: none">• 次要錯誤修正。
iOS 廣播 SDK 1.15.0	<p>適用於即時串流的下載：https://broadcast.live-video.net/1.15.0/AmazonIVSBroadcast-Stages.xcframework.zip</p> <p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.15.0/ios</p> <ul style="list-style-type: none">• 新增了 AVPictureInPicture Controller 延伸模組，以允許使用 IVSImagePreviewView 建立新的執行個體。• 在 IVSImageDevice 上新增了 API，以建立 AVSampleBufferDisplayLayer，供裝置轉譯至其中。• 修正了執行 iOS 17 及更新版本的裝置上發生的低位元速率問題。• 次要錯誤修正。

廣播 SDK 大小：Android

架構	壓縮大小	未壓縮大小
arm64-v8a	5.243 MB	13.194 MB
armeabi-v7a	4.541 MB	9.188 MB
x86_64	5.628 MB	14.455 MB
x86	5.434 MB	14.046 MB

廣播 SDK 大小：iOS

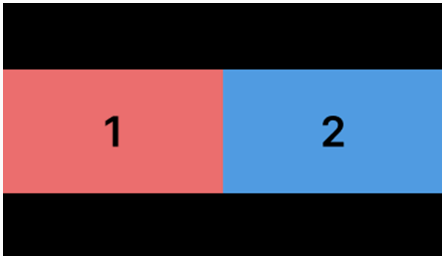
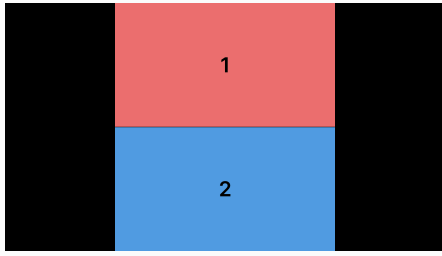
架構	壓縮大小	未壓縮大小
arm64	3.358 MB	7.820 MB

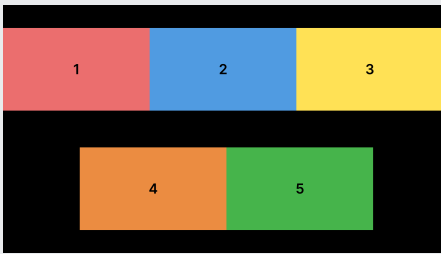


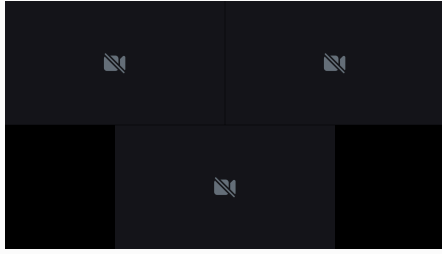
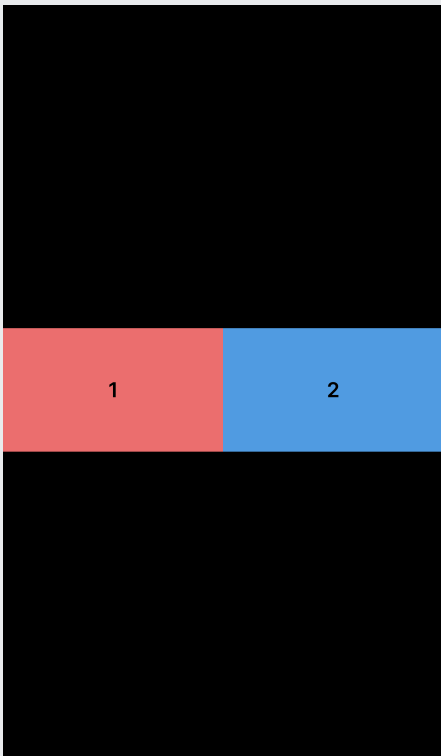
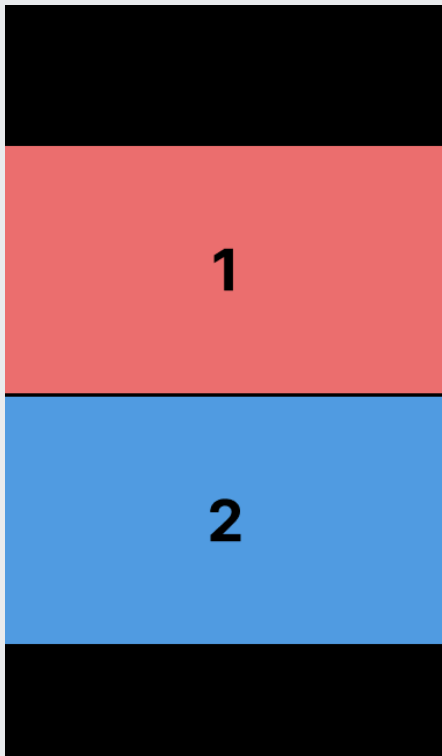
2024 年 2 月 7 日

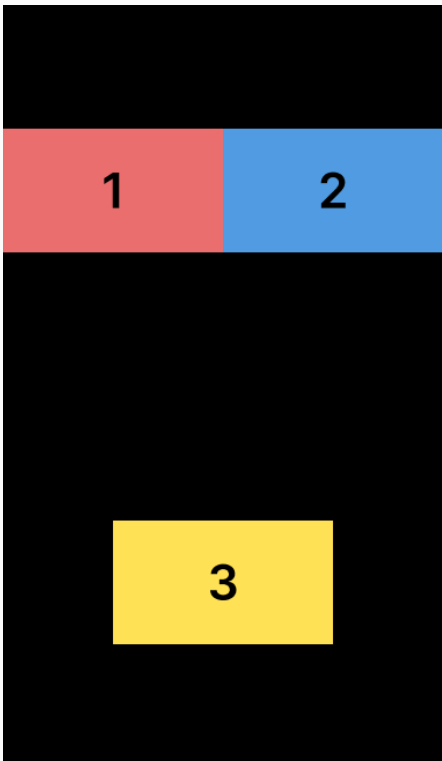
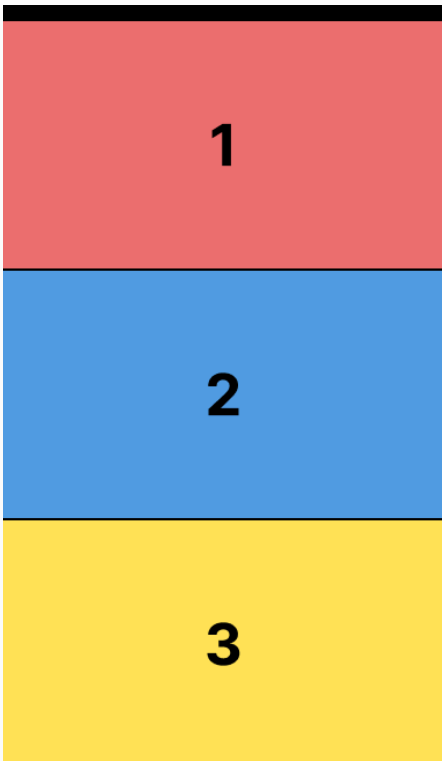
伺服器端合成版面配置更新

此版本對預設網格版面配置進行了視覺效果改善。這些變更會最佳化影片的顯示方式，減少空白。這些變更將於 2024 年 3 月 7 日生效。

重要事項：確保應用程式不依賴目前版面配置的特定功能，例如動態磚的大小和位置。您可以隨時對版面配置進行視覺化改善。

變更說明	舊	新增
自動選取參與者的最佳放置位置，以最大化影片大小。		

變更說明	舊	新增
<p>減少間距並最小化黑色長條來提高空間使用率。</p>		
<p>新增「攝影機關閉」指示器，以清楚看到未共用影片的參與者。</p>		
<p>改善縱向使用案例的空間使用率和比例。</p>		

變更說明	舊	新增
<p>減少參與者之間間距並減少上下黑邊或左右黑邊，提高縱向使用案例中的空間使用率。</p>	 <p>The diagram shows a vertical rectangle with a black background. At the top is a black bar. Below it are two boxes: a red box labeled '1' on the left and a blue box labeled '2' on the right. Below these is another black bar. At the bottom center is a yellow box labeled '3'. There is a significant amount of black space around the boxes.</p>	 <p>The diagram shows a vertical rectangle with a black background. It is divided into three stacked colored boxes: a red box labeled '1' at the top, a blue box labeled '2' in the middle, and a yellow box labeled '3' at the bottom. The black space is reduced compared to the old layout, particularly at the top and bottom.</p>

2024 年 2 月 6 日

OBS 和 WHIP 支援

IVS 可與 OBS 等 WHIP 相容編碼器搭配使用，以發布至 IVS 即時串流。WHIP (WebRTC-HTTP 擷取通訊協定) 是為了標準化 WebRTC 擷取而開發的 IETF 草案。請參閱與 [OBS 和 WHIP 支援](#) 有關的新頁面。

2024 年 2 月 1 日

Amazon IVS 廣播 SDK : Android 1.14.1、iOS 1.14.1、Web 1.8.0 (即時串流)

平台	下載與變更
Web 廣播 SDK 1.8.0	<p>參考文件：https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference</p> <ul style="list-style-type: none">• Simulcast 分層編碼功能目前預設為停用。• 修正了以下問題：如果舞台遭刪除，或參與者與伺服器中斷連線，舞台執行個體無法正常中斷連線。SDK 現在會發出狀態為 DISCONNECTED（而不是先 ERRORED，然後是 CONNECTING）的 STAGE_CONNECTION_STATE_CHANGED 事件。• 修正了以下問題：使用空白音訊或影片播放軌更新策略時，發生發布失敗的情形。
Android 廣播 SDK 1.14.1	<p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.14.1/android</p> <ul style="list-style-type: none">• Simulcast 分層編碼功能目前預設為停用。• 將 libWebRTC 從 M108 更新為了 M119。• 修正了數個當機情況，以改善整體穩定性。• 新增了對立體聲發布的支援。可以透過 StageAudioConfiguration 物件啟用此功能。• 修正了以下問題：參與者在加入工作階段後，出現黑色摘要的情形。• 更新了內部 libWebRTC 參考，以避免在相同主機應用程式中包含其他 libWebRTC 版本時發生符號衝突。

平台	下載與變更
iOS 廣播 SDK 1.14.1	<p>適用於即時串流的下載：https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip</p> <p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.14.1/ios</p> <ul style="list-style-type: none"> • Simulcast 分層編碼功能目前預設為停用。 • 將 libWebRTC 從 M108 更新為了 M119。 • 修正了數個當機情況，以改善整體穩定性。 • 新增了對立體聲發布的支援。可以透過 <code>IVSLocalStageStreamAudioConfiguration</code> 啟用此功能。 • 修正了以下錯誤：為其他參與者啟用純音訊模式時發生當機情況。 • 改善了 TTV 並降低了二進位大小。

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 SDK：Android 1.13.4、iOS 1.13.4、Web 1.7.0 (即時串流)

平台	下載與變更
Web 廣播 SDK 1.7.0	<p>參考文件：https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference</p> <ul style="list-style-type: none"> • 改善了訂閱用戶加入舞台的影片播放時間。 • 移除了 minAudioBitrateKbps 屬性 (未使用)。 • 改善了網際網路中斷或變更期間的網路復原。
Android 廣播 SDK 1.13.4	<p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.13.4/android</p> <ul style="list-style-type: none"> • StageAudioConfiguration 現在支援設定是否應啟用回音消除。
iOS 廣播 SDK 1.13.4	<p>適用於即時串流的下載：https://broadcast.live-video.net/1.13.4/AmazonIVSBroadcast-Stages.xcframework.zip</p> <p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.13.4/ios</p> <ul style="list-style-type: none"> • 在 iOS 上，我們同時改善了錄製和播放的音訊引擎，提升了穩定性和復原性。這可增強對

平台	下載與變更
	<p>使用中路由變更的支援、改善邊緣案例的電池恢復，並減少主執行緒封鎖量。</p> <ul style="list-style-type: none"> 修正了以下問題：即使麥克風已與舞台分離，仍可能保持作用中狀態，導致 iOS 隱私權指示器保持亮起。(SDK 當時未處理傳入音訊。)

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

新的 CloudWatch 指標

我們將 PacketLoss (Stage) 指標重新命名為 DownloadPacketLoss (Stage)。我們還發布了 IVS 即時串流的其他 CloudWatch 指標：

- DownloadPacketLoss (Stage,Participant)
- DroppedFrames (Stage,Participant)

- `SubscribeBitrate` (Stage,Participant,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)	<ul style="list-style-type: none"> • 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	<p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.13.2/android</p> <ul style="list-style-type: none"> • 改善了加入工作階段的第一個階段載入影片 (TTV) 所需的時間。
iOS 廣播 SDK 1.13.2	<p>適用於即時串流的下載：https://broadcast.live-video.net/1.13.2/AmazonIVSBroadcastStages.xcframework.zip</p> <p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.13.2/ios</p> <ul style="list-style-type: none"> • 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	參考文件： https://aws.github.io/amazon-ivs-broadcast-docs/1.13.1/android <ul style="list-style-type: none"> 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)	<ul style="list-style-type: none"> • 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	<p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.13.0/android</p> <ul style="list-style-type: none"> • 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
	<p>回 <code>TextureView</code> 的子類別，但這可能會在未來的 SDK 版本中發生變更。</p> <ul style="list-style-type: none"> • 如果應用程式具體取決於 <code>TextureView</code>，您可以繼續執行而不進行任何變更。您也可以從 <code>getPreview</code> 切換到 <code>getPreviewTextureView</code>，為預設 <code>getPreview</code> 傳回的最終變更做好準備。 • 如果應用程式未明確需要 <code>TextureView</code>，建議您切換為 <code>getPreviewSurfaceView</code> 來降低 CPU 和記憶體使用量。 • SDK 現在實作了一種名為 <code>ImagePreviewSurfaceTarget</code> 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 <code>onFrame</code> 回呼的情況。 • <code>SurfaceSource # getInputSurface</code> 現在加上了 <code>@Nullable</code> 註釋。您的代碼應該在使用之前加以檢查。 • 新增了 <code>UserId</code> 和 <code>attributes</code> 至 <code>ParticipantInfo</code>。<code>UserId</code> 和 <code>attributes</code> 屬性內嵌在權杖中，而應用程式可以在參與者加入時透過 <code>ParticipantInfo</code> 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 <code>StageVideoConfiguration # setCameraCaptureQuality</code> 調整解析度和/或影格率 (fps)。

平台	下載與變更
<p>iOS 廣播 SDK 1.13.0</p>	<p>下載與變更</p> <ul style="list-style-type: none"> 設定組態屬性時拋出的 <code>IllegalArgumentException</code> 現在在例外狀況訊息中包含提供的值。 <p>適用於即時串流的下載：https://broadcast.live-video.net/1.13.0/AmazonIVSBroadcast-Stages.xcframework.zip</p> <p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.13.0/ios</p> <ul style="list-style-type: none"> 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 <code>libWebRTC</code> 的其他程式庫的應用程式不會再與 IVS 廣播 SDK 發生衝突。 <code>IVSStageRenderer</code> 通訊協定上的所有方法現在都會標記為 <code>@optional</code>。 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 <code>isDefault</code> 屬性值可為 <code>true</code>，由作業系統決定的每個位置各一個。 新增了 <code>IVSStageAudioManager</code>，可以精確控制基礎 <code>AVAudioSession</code>，為階段功能提供更廣泛的使用案例。 已新增 <code>UserId</code> 到 <code>ParticipantInfo</code>。

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、S3DestinationConfiguration、S3Detail、S3StorageConfiguration)；我們也修改了 3 個物件 (Composition、Destination、DestinationConfiguration)；這會影響 GetComposition 回應，以及 StartComposition 請求和回應。

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。
- [搭配 IVS 即時串流使用 Amazon EventBridge](#)；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 Composition 和 EncoderConfiguration 端點以及 11 個物件 (ChannelDestinationConfiguration、Composition、CompositionSummary、Destination、DestinationConfiguration 和 Video)。

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	<p>參考文件：https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference</p> <ul style="list-style-type: none">• 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

新的 CloudWatch 指標和參與者資料

我們發布 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 `GetParticipant` 回應。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	<p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.12.1/android</p> <ul style="list-style-type: none">修正了呼叫 <code>BroadcastSession.setListener</code> 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	<p>參考文件：https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference</p> <ul style="list-style-type: none"> 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	<p>參考文件：https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference</p> <ul style="list-style-type: none"> 修正了 TypeScript 5 上的內部 Maybe 類型錯誤。 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)	<ul style="list-style-type: none"> 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。

平台	下載與變更
Android 廣播 SDK 1.12.0	<ul style="list-style-type: none"> 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。 在 ImageDevice 上加入了新的 <code>setOnFrameCallback</code> 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。 <p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.12.0/android</p> <ul style="list-style-type: none"> 現在已支援 Android 9。 改善 CPU 使用率和效能。
iOS 廣播 SDK 1.12.0	<p>適用於即時串流的下載：https://broadcast.live-video.net/1.12.0/AmazonIVSBroadcast-Stages.xcframework.zip</p> <p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.12.0/ios</p> <ul style="list-style-type: none"> 更正了 <code>IVSDeviceDiscovery.createAudioSourceWithName</code> 的簽名以傳回 <code>IVSCustomAudioSource</code> 而不是 <code>IVSCustomImageSource</code>。

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

架構	壓縮大小	未壓縮大小
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	<p>參考文件：https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-reference</p> <ul style="list-style-type: none"> 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)	<p>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」。</p>
Android 廣播 SDK 1.11.0	<p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.11.0/android</p> <ul style="list-style-type: none"> 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。)

平台	下載與變更
iOS 廣播 SDK 1.11.0	<p>適用於即時串流的下載：https://broadcast.live-video.net/1.11.0/AmazonIVSBroadcast-Stages.xcframework.zip</p> <p>參考文件：https://aws.github.io/amazon-ivs-broadcast-docs/1.11.0/ios</p> <ul style="list-style-type: none"> 更正了 <code>IVSDeviceDiscovery.createAudioSourceWithName</code> 的簽名以傳回 <code>IVSCustomAudioSource</code> 而不是 <code>IVSCustomImageSource</code>。

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](#)。