aws

실시간 스트리밍 사용 설명서

Amazon IVS

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

Amazon IVS: 실시간 스트리밍 사용 설명서

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

Amazon의 상표 및 트레이드 드레스는 Amazon이 제공하지 않는 제품 또는 서비스와 관련하여 고객에게 혼동을 유발할 수 있는 방식 또는 Amazon을 폄하하거나 평판에 악영향을 주는 방식으로 사용될 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon 계열사, 관련 업체 또는 Amazon의 지원 업체 여부에 상관없이 해당 소유자의 자산입니다.

Table of Contents

Amazon IVS 실시간 스트리밍이란?

Amazon Interactive Video Service(IVS) 실시간 스트리밍은 애플리케이션에 실시간 오디오와 비디오를 추가하는 데 필요한 모든 것을 제공합니다.

장점:

- 실시간 지연 시간 - 지연 시간에 민감한 사용 사례를 위한 애플리케이션을 구축하여 시청자가 IVS 실시간 스트리밍을 통해 연결 상태를 유지하고 참여할 수 있도록 지원합니다. 호스트에서 시청자까지 300밀리초 미만의 지연 시간으로 라이브 스트리밍을 제공합니다.
- 높은 동시 접속자 수 - IVS 실시간 스트리밍으로 대규모 상호 작용의 잠재력을 활용합니다. 최대 1만 명의 시청자를 수용하고 최대 12개의 호스트가 가상 무대에 오를 수 있도록 지원합니다.
- 모바일 최적화 - IVS 실시간 스트리밍은 모바일 사용 사례에 최적화되어 다양한 디바이스 및 네트워크 기능을 제공합니다. Android 및 iOS용 Amazon IVS 브로드캐스트 SDK를 통합하면 사용자가 호스트 또는 시청자로 참여하여 모바일 디바이스에서 고품질 라이브 스트림을 즐길 수 있습니다.

사용 사례:

- 게스트 스팟 - 호스트가 '스테이지에서' 게스트를 홍보할 수 있는 애플리케이션을 생성하고 시청자가 호스트가 되어 실시간 상호작용을 할 수 있습니다.
- 비교(VS) 모드 - 동시 경쟁을 통한 경험을 제공하고 시청자가 실시간으로 호스트의 경쟁을 시청할 수 있도록 지원합니다.
- 오디오 룸 - 청취자를 게스트로 대화에 초대하여 오디오 룸에서 더 깊은 참여를 유도할 수 있습니다.
- 실시간 비디오 경매 - 경매를 양방향 비디오 이벤트로 전환하고 실시간 지연 시간으로 경매의 흥미와 무결성을 유지합니다..

여기에 있는 제품 설명서 외에도 게시된 콘텐츠(데모, 코드 샘플, 블로그 게시물)를 검색하고, 비용을 예측하고, 라이브 데모를 통해 Amazon IVS를 경험할 수 있는 전용 사이트인 https://ivs.rocks/를 참조하세요.

글로벌 솔루션, 리전별 제어

글로벌 스트리밍 및 보기

Amazon IVS를 사용하여 전 세계 시청자에게 스트리밍할 수 있습니다.

- 스트리밍할 때 Amazon IVS는 사용자와 가까운 위치에서 자동으로 비디오를 수집합니다.
- 시청자는 전 세계에서 실시간 스트림을 시청할 수 있습니다.

다시 말해서, '데이터 플레인'이 글로벌하다는 것입니다. 데이터 플레인은 스트리밍/인제스트 및 보기를 의미합니다.

리전별 제어

Amazon IVS 데이터 플레인은 글로벌이지만 '컨트롤 플레인'은 리전별입니다. 컨트롤 플레인은 Amazon IVS 콘솔, API 및 리소스(스테이지)를 말합니다.

이를 Amazon IVS는 '리전별 AWS 서비스'라고 표현할 수도 있습니다. 즉, 각 리전에 있는 Amazon IVS 리소스는 다른 리전에 있는 유사한 리소스와 독립되어 있습니다. 예를 들어, 한 리전에서 생성한 스테이지는 다른 리전에서 생성한 스테이지와 독립적입니다.

리소스를 사용할 때(예: 스테이지 생성) 생성할 리전을 지정해야 합니다. 그런 다음, 리소스를 관리할 때는 리소스가 생성된 리전과 동일한 리전에서 관리해야 합니다.

사용하는 도구...	리전을 지정하는 방법...
Amazon IVS 콘솔	탐색 모음 오른쪽 상단에 있는 [리전 선택(Select a Region)] 드롭다운을 사용합니다.
Amazon IVS API	적절한 서비스 엔드포인트를 사용합니다. Amazon IVS Real-Time Streaming API Reference를 참조하세요. (SDK를 통해 API에 액세스하는 경우 SDK의 `region` 파라미터를 설정합니다. AWS 기반 구축 도구를 참조하세요.)
AWS CLI	다음 중 하나를 사용합니다. • CLI 명령에 `--region <aws-region>` 을 추가합니다. • 로컬 AWS 구성 파일에 리전을 입력합니다.

스테이지가 생성된 리전과 상관없이 어디에서나 Amazon IVS로 스트리밍할 수 있으며 시청자는 어디에서나 시청할 수 있습니다.

IVS 실시간 스트리밍 시작하기

이 문서에서는 Amazon IVS 실시간 스트리밍을 앱에 통합하는 단계를 안내합니다.

주제

소개

사전 조건

실시간 스트리밍을 처음 사용하는 경우, 먼저 다음 작업을 완료해야 합니다. 지침은 IVS 지연 시간이 짧은 스트리밍 시작하기를 참조하세요.

- AWS 계정 생성
- 루트 및 관리 사용자 설정

기타 참조

- IVS Web Broadcast SDK Reference
- IVS Android Broadcast SDK Reference
- IVS iOS Broadcast SDK Reference
- IVS Real-Time Streaming API Reference

실시간 스트리밍 용어

용어	설명
단계	참가자들이 실시간으로 비디오를 교환할 수 있는 가상 공간입니다.
Host	스테이지로 로컬 비디오를 전송하는 참가자.
뷰어	호스트의 비디오를 수신하는 참가자.
Participant	스테이지에 호스트 또는 시청자로 연결된 사용자입니다.
참가자 토큰	스테이지에 참가할 때 참가자를 인증하는 토큰입니다.
브로드캐스트 SDK	참가자가 비디오를 전송하고 수신할 수 있도록 하는 클라이언트 라이브러리입니다.

단계 개요

1. the section called “IAM 권한 설정” - 사용자에게 기본 권한 세트를 제공하는 AWS Identity and Access Management(IAM) 정책을 생성하고 해당 정책을 사용자에게 할당합니다.
2. 스테이지 생성 - 참가자들이 실시간으로 비디오를 교환할 수 있는 가상 공간을 생성합니다.
3. 참가자 토큰 배포 - 참가자가 스테이지에 참가할 수 있도록 토큰을 전송합니다.
4. IVS 브로드캐스트 SDK 통합 - 참가자가 비디오를 전송하고 수신할 수 있도록 앱에 브로드캐스트 SDK를 추가합니다(the section called “웹”, the section called “Android” 및 the section called “iOS”).
5. 비디오 게시 및 구독 - 비디오를 스테이지로 전송하고 다른 호스트로부터 비디오를 수신합니다(the section called “웹”, the section called “Android” 및 the section called “iOS”).

IAM 권한 설정

다음으로 사용자에게 기본 권한 집합(예: Amazon IVS 스테이지 생성 및 참가자 토큰 생성)을 제공하는 AWS Identity and Access Management(IAM) 정책을 생성하고 해당 정책을 사용자에게 할당해야 합니다. 새 사용자를 생성할 때 권한을 할당하거나 기존 사용자에 권한을 추가할 수 있습니다. 두 절차 모두 아래에 나와 있습니다.

IAM 사용자 및 정책에 대한 자세한 내용, 사용자에게 정책을 연결하는 방법, Amazon IVS로 수행할 수 있는 작업 등 자세한 내용은 다음을 참조하세요.

- IAM 사용 설명서의 IAM 사용자 생성
- IAM의 Amazon IVS 보안 및 'IVS에 대한 관리형 정책'의 정보입니다.
- Amazon IVS 보안의 IAM 정보

Amazon IVS에 대한 기존 AWS 관리형 정책을 사용하거나 사용자, 그룹 또는 역할 집합에 부여할 권한을 사용자 지정하는 새 정책을 생성할 수 있습니다. 아래에 두 가지 접근 방식이 모두 설명되어 있습니다.

IVS 권한에 대한 기존 정책 사용

대부분의 경우 Amazon IVS에 대한 AWS 관리형 정책을 사용하는 것이 좋습니다. IVS 보안의 IVS에 대한 관리형 정책 섹션에 자세히 설명되어 있습니다.

- `IVSReadOnlyAccess` AWS 관리형 정책을 사용하여 애플리케이션 개발자에게 모든 IVS Get 및 List API 엔드포인트에 대한 액세스 권한을 부여합니다(저지연 스트리밍과 실시간 스트리밍에 모두 해당).
- `IVSFullAccess` AWS 관리형 정책을 사용하여 애플리케이션 개발자에게 모든 IVS API 엔드포인트에 대한 액세스 권한을 부여합니다(저지연 스트리밍과 실시간 스트리밍에 모두 해당).

선택 사항: Amazon IVS 권한에 대한 사용자 지정 정책 생성

다음 단계를 수행합니다.

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 정책을 선택한 다음 정책 생성을 선택합니다. 권한 지정 창이 열립니다.

3. 권한 지정 창에서 JSON 탭을 선택하고 다음 IVS 정책을 복사하여 권한 편집기 텍스트 영역에 붙여 넣습니다. (일부 Amazon IVS 작업은 정책에 포함되지 않습니다. 필요에 따라 엔드포인트 액세스 권한을 추가/삭제(허용/거부)할 수 있습니다. IVS 엔드포인트에 대한 자세한 내용은 IVS Real-Time Streaming API Reference를 참조하세요.)

```
{
   "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. 여전히 권한 지정 창에서 다음(창 아래쪽으로 스크롤하여 확인)을 선택합니다. 검토 및 생성 창이 열립니다.
5. 검토 및 생성 창에서 정책 이름을 입력하고 선택적으로 설명을 추가합니다. 사용자를 생성할 때(아래) 필요한 정책 이름을 기록해 둡니다. 정책 생성(창 하단)을 선택합니다.
6. IAM 콘솔 창으로 돌아가면 새 정책이 생성되었음을 확인하는 배너가 표시됩니다.

새 사용자 생성 및 권한 추가

IAM 사용자 액세스 키

IAM 액세스 키는 액세스 키 ID와 비밀 액세스 키로 구성됩니다. 이 키들은 AWS에 보내는 프로그래밍 방식의 요청에 서명하는 데 사용됩니다. 액세스 키가 없는 경우 AWS Management Console에서 액세스 키를 생성할 수 있습니다. 루트 사용자 액세스 키는 사용하지 않는 것이 좋습니다.

액세스 키를 생성할 때에 한하여 비밀 액세스 키를 보고 다운로드할 수 있습니다. 나중에 복구할 수 없습니다. 언제든지 새 액세스 키를 생성할 수 있지만 필요한 IAM 작업을 수행할 수 있는 권한이 있어야 합니다.

항상 액세스 키를 안전하게 보관하세요. 절대로 (Amazon에서 문의가 온 것처럼 보여도) 제3자와 공유하지 마세요. 자세한 내용은 IAM 사용 설명서의 IAM 사용자의 액세스 키 관리를 참조하세요.

절차

다음 단계를 따릅니다:

1. 탐색 창에서 사용자를 선택한 다음, 사용자 생성을 선택합니다. 사용자 세부 정보 지정 창이 열립니다.
2. 사용자 세부 정보 지정 창에서,
   a. 사용자 세부 정보에서 생성하려고 하는 신규 사용자 이름을 입력합니다.
   b. AWS Management Console에 대한 사용자 액세스 제공을 선택합니다.
   c. 콘솔 암호에서 자동 생성된 암호(권장)을 선택합니다.
   d. 다음 로그인 시 사용자가 새 암호를 생성해야 함을 선택합니다.
   e. 다음을 선택합니다. 권한 설정 창이 열립니다.

3. 권한 설정에서 정책 직접 연결을 선택합니다. 권한 정책 창이 열립니다.
4. 검색 상자에 IVS 정책 이름을 입력합니다(AWS 관리형 정책 또는 이전에 생성한 사용자 지정 정책). 검색되면 확인란을 선택하여 정책을 선택합니다.
5. 다음(창 하단)을 선택합니다. 검토 및 생성 창이 열립니다.
6. 검토 및 생성 창에서 모든 사용자 세부 정보가 올바른지 확인한 다음 사용자 생성(창 하단)을 선택합니다.
7. 콘솔 로그인 세부 정보가 포함되어 있는 암호 검색 창이 열립니다. 해당 정보를 안전하게 저장하여 향후에 참조하세요. 완료되면 사용자 목록으로 돌아가기를 선택하세요.

기존 사용자에 권한 추가

다음 단계를 수행합니다.

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 사용자를 선택한 다음 업데이트할 기존 사용자 이름을 선택합니다. (이름을 클릭하여 선택합니다. 선택 상자는 선택하지 마세요.)
3. 요약 페이지의 권한 탭에서 권한 추가를 선택합니다. 권한 추가 창이 열립니다.
4. 기존 정책 직접 연결을 선택합니다. 권한 정책 창이 열립니다.
5. 검색 상자에 IVS 정책 이름을 입력합니다(AWS 관리형 정책 또는 이전에 생성한 사용자 지정 정책). 정책을 찾으면 확인란을 선택하여 정책을 선택합니다.
6. 다음(창 하단)을 선택합니다. 검토 창이 열립니다.
7. 검토 창에서 권한 추가(창 하단)를 선택합니다.
8. 요약 페이지에서 IVS 정책이 추가되었는지 확인합니다.

스테이지 생성

스테이지는 참가자들이 실시간으로 비디오를 교환할 수 있는 가상 공간입니다. 실시간 스트리밍 API의 기본 리소스입니다. 콘솔이나 CreateStage 엔드포인트를 사용하여 스테이지를 생성할 수 있습니다.

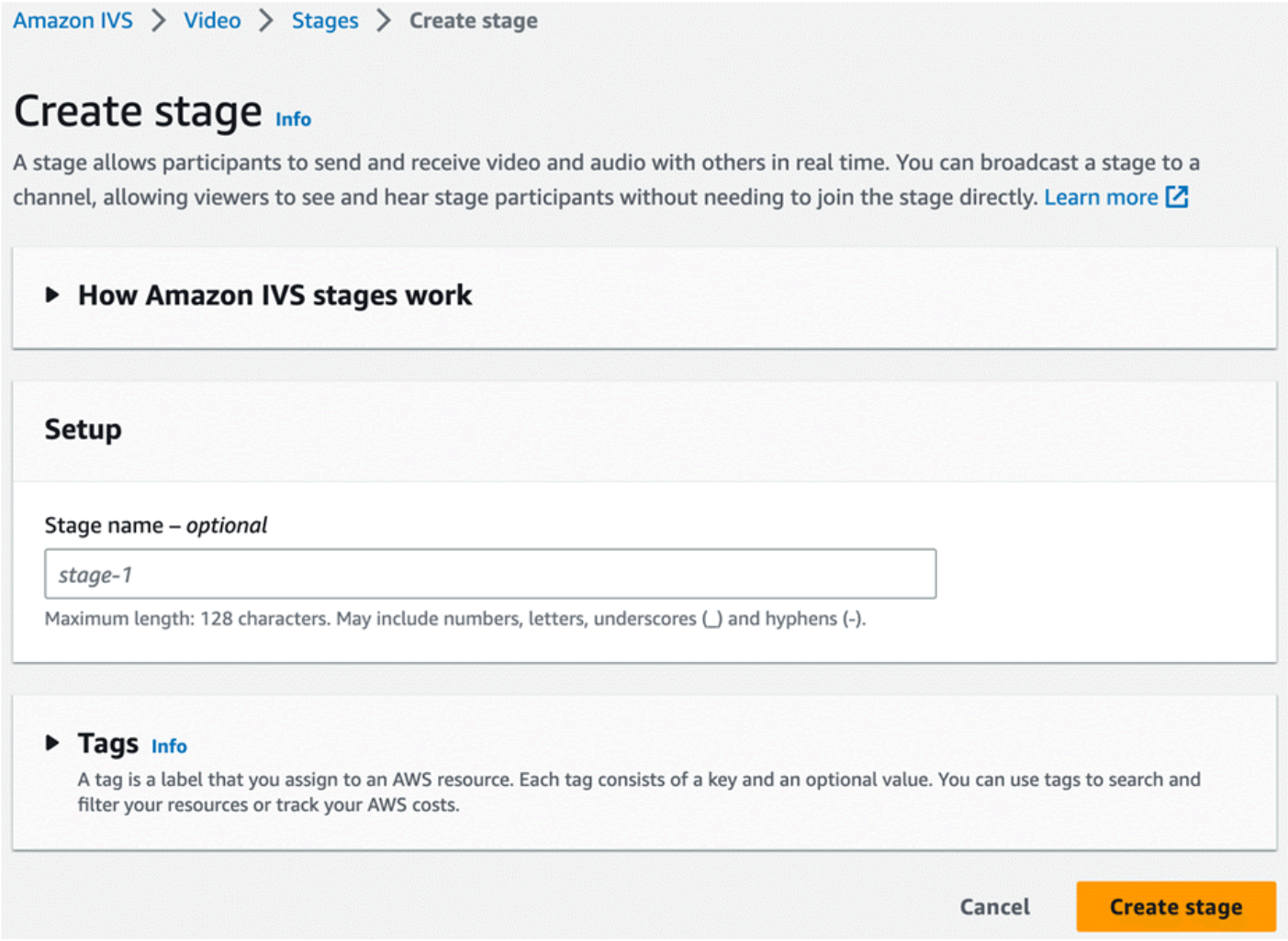
가능한 경우 재사용을 위해 이전 스테이지를 유지하는 대신 각 논리적 세션에 대해 새 스테이지를 생성하고 완료되면 삭제하는 것이 좋습니다. 기한 경과 리소스(재사용하지 않는 오래된 스테이지)를 정리하지 않으면 최대 스테이지 수 제한에 더 빨리 도달할 수 있습니다.

콘솔 지침

1. Amazon IVS 콘솔을 엽니다.

   (또는 AWS Management Console을 통해서 Amazon IVS 콘솔에 액세스할 수 있습니다.)

2. 왼쪽 탐색 창에서 스테이지를 선택한 다음 스테이지 생성을 선택합니다. 스테이지 생성 창이 나타납니다.



3. 필요에 따라 스테이지 이름을 입력합니다. 스테이지 생성을 선택하여 스테이지를 생성합니다. 새 스테이지의 스테이지 세부 정보 페이지가 나타납니다.

CLI 지침

AWS CLI를 설치하려면 AWS CLI의 최신 버전 설치 또는 업데이트를 참조하세요.

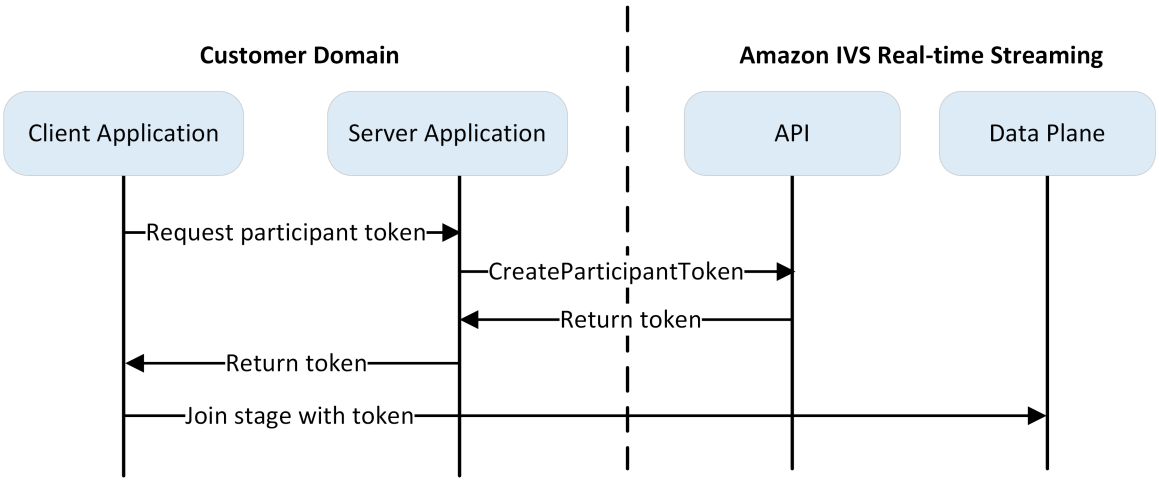
이제 CLI를 사용하여 리소스를 생성하고 관리할 수 있습니다. 스테이지 API는 ivs-realtime 네임스페이스 아래에 있습니다. 예를 들어, 스테이지를 생성하려면:

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

다음과 같이 응답합니다.

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

참가자 토큰 배포



이제 스테이지가 있으므로 참가자가 스테이지에 참가하고 비디오 전송 및 수신을 시작할 수 있도록 토큰을 생성하고 참가자에게 배포해야 합니다.

위에 표시된 것처럼 클라이언트 애플리케이션은 서버 애플리케이션에 토큰을 요청하고 서버 애플리케이션은 AWS SDK 또는 SigV4 서명 요청을 CreateParticipantToken 사용하여 호출합니다. AWS 자격 증명은 API를 직접적으로 호출하는 데 사용되므로 토큰은 클라이언트 측 애플리케이션이 아닌 안전한 서버 측 애플리케이션에서 생성되어야 합니다.

참가자 토큰을 생성할 때 해당 토큰에 의해 활성화되는 기능을 선택적으로 지정할 수 있습니다. 기본값은 참가자가 오디오와 비디오를 전송하고 수신을 수 있도록 하는 PUBLISH 및 SUBSCRIBE이지만 기능의 하위 세트로 토큰을 발행할 수 있습니다. 예를 들어 진행자를 위한 SUBSCRIBE 기능만 있는 토큰을

발급할 수 있습니다. 이 경우 진행자는 비디오를 전송하는 참가자를 볼 수 있지만 직접 비디오를 전송할 수는 없습니다.

테스트 및 개발을 위해 콘솔이나 CLI를 통해 참가자 토큰을 생성할 수 있지만, 프로덕션 환경에서는 AWS SDK를 사용하여 생성하는 것이 가장 좋습니다.

서버에서 각 클라이언트로 토큰을 배포하는 방법(예: API 요청을 통해)이 필요합니다. 이 기능은 제공하지 않습니다. 이 가이드에서는 다음 단계에 따라 토큰을 복사하여 클라이언트 코드에 붙여넣기만 하면 됩니다.

중요: 토큰을 불투명한 것으로 취급하세요. 즉, 토큰 콘텐츠를 기반으로 기능을 빌드하지 마세요. 토큰 형식은 향후 변경될 수 있습니다.

콘솔 지침

1. 이전 단계에서 생성한 스테이지로 이동합니다.
2. 참가자 토큰 생성(Create a participant token)을 선택합니다 참가자 토큰 생성 창(Create a participant token)이 나타납니다.
3. 토큰과 연결할 사용자 ID를 입력합니다. UTF-8로 인코딩된 텍스트일 수 있습니다.
4. 참가자 토큰 생성(Create a participant token)을 선택합니다.
5. 토큰을 복사합니다. 중요: 토큰을 저장해야 합니다. IVS는 토큰을 저장하지 않으며 나중에 검색할 수 없습니다..

CLI 지침

AWS CLI를 사용하여 채팅 토큰을 생성하려면 먼저 시스템에 CLI를 다운로드하고 구성해야 합니다. 자세한 내용은 AWS 명령줄 인터페이스 사용 설명서를 참조하세요. AWS CLI를 사용하여 토큰을 생성하는 것은 테스트 목적에 적합하지만, 프로덕션 용도의 경우 AWS SDK를 사용하여 서버 측에서 토큰을 생성하는 것이 좋습니다(아래 지침 참조).

1. 스테이지 ARN과 함께 `create-participant-token` 명령을 실행합니다. `"PUBLISH"`, `"SUBSCRIBE"` 기능 중 일부 또는 전부를 포함합니다.

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

2. 참가자 토큰이 반환됩니다.

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

3. 이 토큰을 저장합니다. 스테이지에 참가하고 비디오를 전송하고 수신하려면 이 토큰이 필요합니다.

AWS SDK 지침

AWS SDK를 사용하여 토큰을 생성할 수 있습니다. 다음은 SDK 사용 지침입니다. AWS JavaScript

중요: 이 코드는 서버 측에서 실행되어야 하며 해당 출력은 클라이언트에 전달되어야 합니다.

전제 조건: 아래 코드 샘플을 사용하려면 aws-sdk/ 패키지를 설치해야 합니다. client-ivs-realtime 자세한 내용은 의 AWS SDK 시작하기를 참조하십시오. JavaScript

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

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

IVS 브로드캐스트 SDK 통합

IVS는 애플리케이션에 통합할 수 있는 웹, Android 및 iOS용 브로드캐스트 SDK를 제공합니다. 브로드캐스트 SDK는 비디오 전송 및 수신에 모두 사용됩니다. 이 섹션에서는 두 명 이상의 참가자가 실시

간으로 상호 작용할 수 있는 간단한 애플리케이션을 작성하는 방법에 대해 설명하고 있습니다. 아래 단계는 이라는 BasicRealTime 앱을 만드는 과정을 안내합니다. 전체 앱 코드가 켜져 CodePen 있으며 GitHub 다음과 같습니다.

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

웹

파일 설정

시작하려면 폴더와 초기 HTML 및 JS 파일을 생성하여 파일을 설정합니다.

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

스크립트 태그 또는 npm을 사용하여 브로드캐스트 SDK를 설치할 수 있습니다. 이 예제에서는 간단히 하기 위해 스크립트 태그를 사용하지만 나중에 npm을 사용하도록 선택하려는 경우 쉽게 수정할 수 있습니다.

스크립트 태그 사용

웹 브로드캐스트 SDK는 JavaScript 라이브러리로 배포되며 https://web-broadcast.live-video.net/1.8.0/amazon-ivs-web-broadcast .js에서 검색할 수 있습니다.

`<script>` 태그를 통해 로드되면 라이브러리는 `IVSBroadcastClient`라는 창 범위에 글로벌 변수를 노출합니다.

npm 사용

npm 패키지 설치

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

이제 IVS 객체에 액세스할 수 있습니다. BroadcastClient

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

Android

Android 프로젝트 생성

1. Android Studio에서 새 프로젝트를 생성합니다.
2. 빈 보기 활동을 선택합니다.

   참고: 일부 이전 버전의 Android Studio에서는 보기 기반 활동을 빈 활동이라고 합니다. Android Studio 창에 빈 활동이 표시되고 빈 보기 활동이 표시되지 않으면 빈 활동을 선택합니다. 그렇지 않으면 Jetpack Composite가 아닌 View API를 사용하므로 빈 활동을 선택하지 마세요.
3. 프로젝트 이름을 지정한 다음 완료를 선택합니다.

브로드캐스트 SDK 설치

Amazon 개발 환경에 Amazon IVS Android 브로드캐스트 라이브러리를 추가하려면 여기에 나온 것처럼(최신 버전의 Amazon IVS 브로드캐스트 SDK용) 라이브러리를 모듈의 `build.gradle` 파일에 추가합니다. 최신 프로젝트에서는 `mavenCentral` 리포지토리가 이미 `settings.gradle` 파일에 포함되어 있을 수 있습니다. 이 경우 `repositories` 블록을 생략할 수 있습니다. 샘플의 경우 `android` 블록에서 데이터 바인딩도 활성화해야 합니다.

```
android {
    dataBinding.enabled true
}

repositories {
    mavenCentral()
}

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

또는 SDK를 수동으로 설치하려면 다음 위치에서 최신 버전을 다운로드하세요.

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

iOS

iOS 프로젝트 생성

1. 새 Xcode 프로젝트를 생성합니다
2. 플랫폼에서 iOS를 선택합니다.
3. 애플리케이션에서 앱을 선택합니다.
4. 앱의 제품 이름을 입력하고 다음을 선택합니다.
5. 프로젝트를 저장할 디렉터리를 선택(이동)하고 생성을 선택합니다.

다음으로 SDK를 가져와야 합니다. 를 통해 브로드캐스트 SDK를 통합하는 것이 좋습니다. CocoaPods 또는 프레임워크를 프로젝트에 수동으로 추가할 수 있습니다. 두 방법 모두 아래에 설명되어 있습니다.

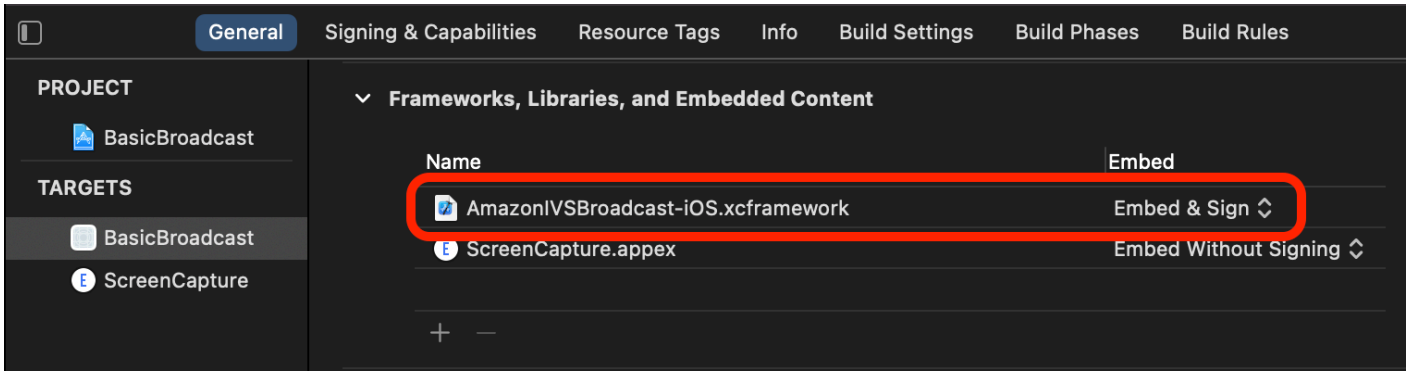
권장 사항: 브로드캐스트 SDK 설치 () CocoaPods

프로젝트 이름이 `BasicRealTime`이라고 가정하고 프로젝트 폴더에 다음과 같은 내용의 `Podfile`을 생성한 다음 `pod install`을 실행합니다.

```
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
```

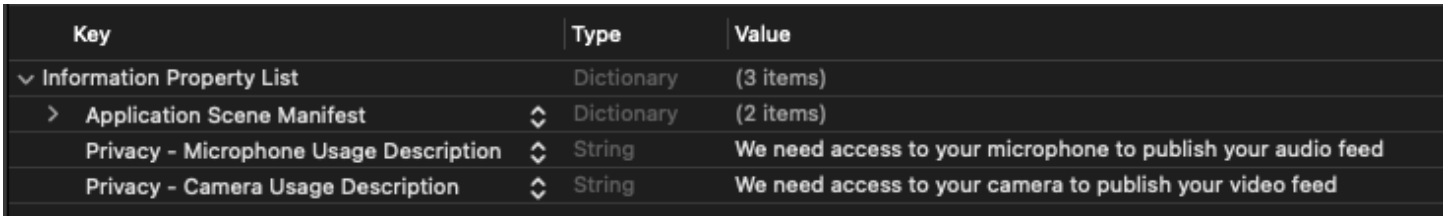
대체 방법: 수동으로 프레임워크 설치

1. https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip 에서 최신 버전을 다운로드하십시오.
2. 아카이브 콘텐츠의 압축을 풉니다. `AmazonIVSBroadcast.xcframework`에는 디바이스와 시뮬레이터 모두에 대한 SDK가 포함되어 있습니다.
3. `AmazonIVSBroadcast.xcframework`를 애플리케이션 대상의 일반(General) 탭의 프레임워크, 라이브러리 및 포함된 콘텐츠(Frameworks, Libraries, and Embedded Content) 섹션으로 드래그하여 포함시킵니다.



권한 구성

프로젝트의 `Info.plist`를 업데이트하여 `NSCameraUsageDescription`과 `NSMicrophoneUsageDescription`에 대한 2개의 새 항목을 추가해야 합니다. 값으로 앱에서 카메라 및 마이크 액세스를 요청하는 이유에 대한 사용자 대면 설명을 제공합니다.



비디오 게시 및 구독

웹, 안드로이드, iOS에 대한 자세한 내용은 아래를 참조하십시오.

웹

HTML 표준 문안 생성

먼저 HTML 표준 문안을 생성하고 라이브러리를 스크립트 태그로 가져오겠습니다.

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

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

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

<body>

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

</body>
</html>
```

토큰 입력 수락 및 Join/Leave 버튼 추가

여기서는 입력 통제로 본문을 채웁니다. 입력 통제는 토큰을 입력으로 사용하고 Join 및 Leave 버튼을 설정합니다. 일반적으로 애플리케이션은 애플리케이션의 API에서 토큰을 요청하지만 이 예제에서는 토큰을 복사하여 토큰 입력에 붙여넣습니다.

```
<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 />
```

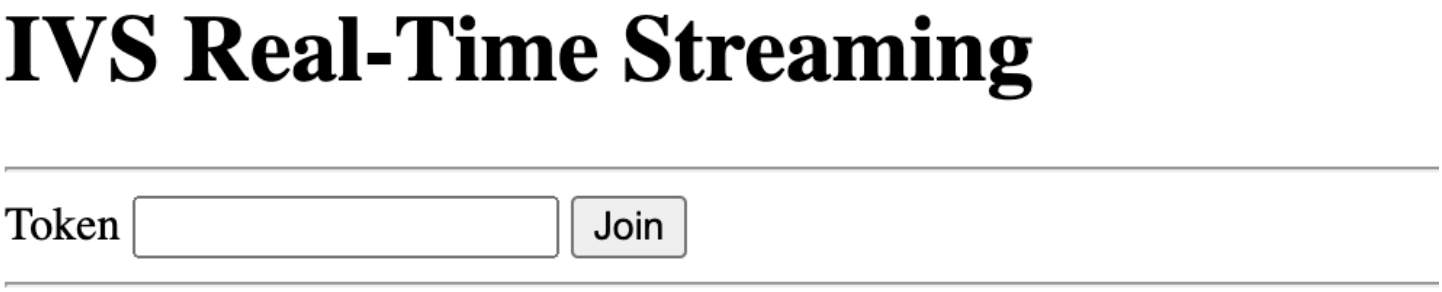
미디어 컨테이너 요소 추가

이러한 요소에는 로컬 및 원격 참가자를 위한 미디어가 포함됩니다. `app.js`에 정의된 애플리케이션 로직을 로드하는 스크립트 태그를 추가합니다.

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

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

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

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

이렇게 하면 HTML 페이지가 완성되고 브라우저에서 `index.html`을 로드할 때 다음이 표시됩니다.



app.js 생성

이제 `app.js` 파일의 내용을 정의하도록 하겠습니다. 먼저 SDK의 글로벌에서 필요한 모든 속성을 가져옵니다.

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

애플리케이션 변수 생성

Join 및 Leave 버튼 HTML 요소에 대한 참조를 포함하고 애플리케이션의 상태를 저장하는 변수를 설정합니다.

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

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

joinStage 1 생성: 함수 정의 및 입력 검증

`joinStage` 함수는 입력 토큰을 가져와서 스테이지에 대한 연결을 생성하고 `getUserMedia`에서 검색된 비디오와 오디오를 게시하기 시작합니다.

먼저 함수를 정의하고 상태 및 토큰 입력을 검증합니다. 다음 몇몇 섹션에서 이 함수를 구체화하겠습니다.

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

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

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

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

joinStage 2 생성: 게시할 미디어 가져오기

다음은 스테이지에 게시될 미디어입니다.

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

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

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

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

joinStage 3 생성: 스테이지 전략 정의 및 스테이지 생성

이 단계 전략은 SDK가 게시할 항목과 구독할 참가자를 결정하는 데 사용하는 결정 로직의 핵심입니다. 함수의 용도에 대한 자세한 내용은 전략을 참조하세요.

이 전략은 간단합니다. 스테이지에 참가한 후 방금 검색한 스트림을 게시하고 모든 원격 참가자의 오디오와 비디오를 구독합니다.

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

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

joinStage 4 생성: 스테이지 이벤트 처리 및 미디어 렌더링

스테이지는 많은 이벤트를 내보냅니다. 페이지에서 미디어를 렌더링하고 제거하려면 `STAGE_PARTICIPANT_STREAMS_ADDED`와 `STAGE_PARTICIPANT_LEFT`를 수신해야 합니다. 이벤트에는 보다 포괄적인 이벤트 세트가 나열됩니다.

여기서는 필요한 DOM 요소를 관리하는 데 도움이 되는 4가지 도우미 함수인 `setupParticipant`, `teardownParticipant`, `createVideoEl` 및 `createContainer`를 생성합니다.

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

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

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

    let streamsToDisplay = streams;

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

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

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


// Helper functions for managing DOM

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

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

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

  return videoEl;
}

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

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

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

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

  return participantContainer;
}
```

joinStage 5 생성: 스테이지에 참가

드디어 스테이지에 참가하여 `joinStage` 함수를 완성해 보겠습니다.

```
try {
```

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

leaveStage 생성

leave 버튼이 간접적으로 호출할 `leaveStage` 함수를 정의합니다.

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

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

입력 이벤트 핸들러 초기화

`app.js` 파일에 마지막 함수를 하나 추가하겠습니다. 이 함수는 페이지가 로드될 때 즉시 간접적으로 호출되고 스테이지 참가 및 탈퇴를 위한 이벤트 핸들러를 설정합니다.

```
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
```

애플리케이션 실행 및 토큰 제공

이 부분에서는 로컬에서 또는 다른 사용자와 웹 페이지를 공유하고, 페이지를 열고, 참가자 토큰을 입력하여 스테이지에 참가할 수 있습니다.

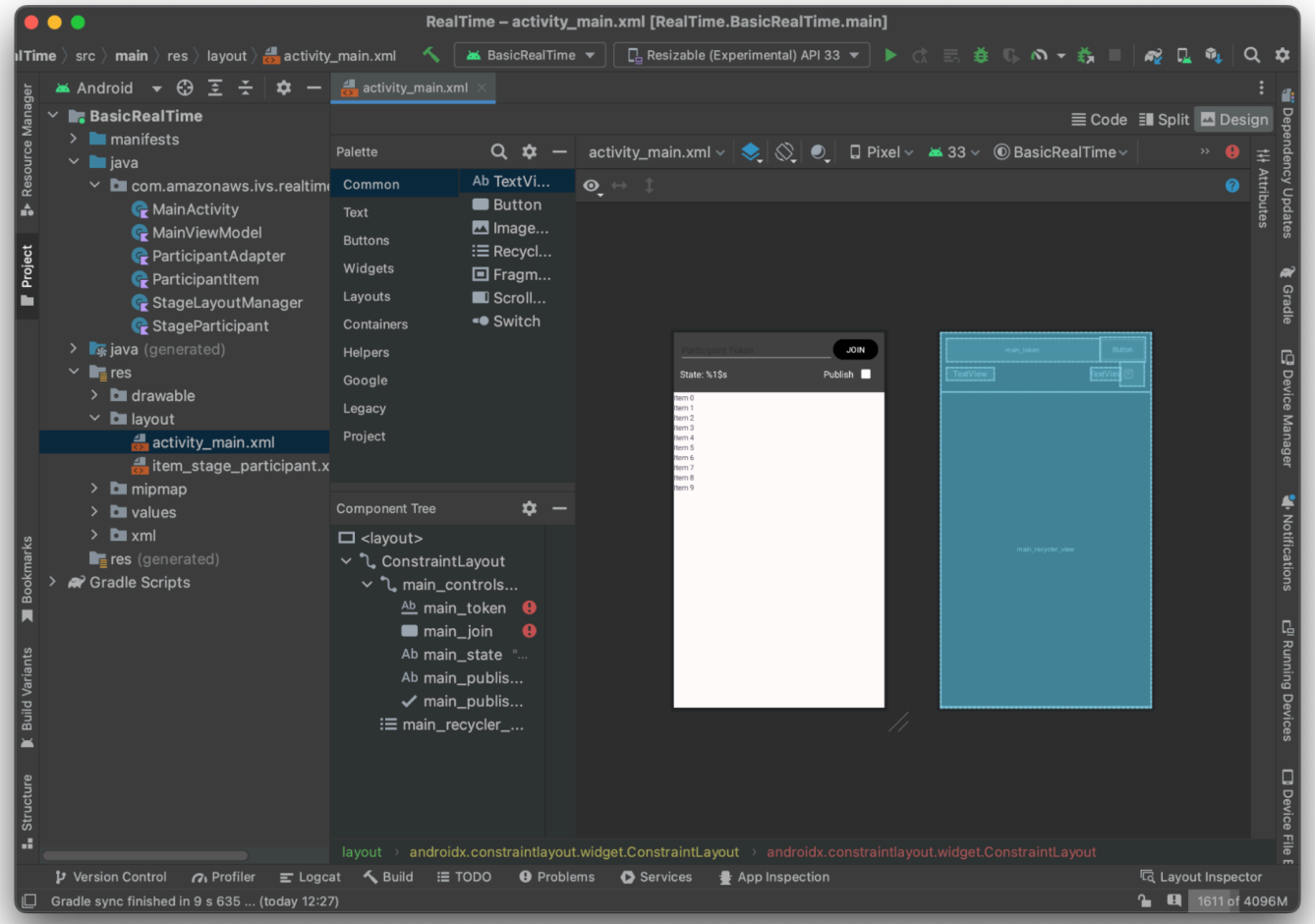
다음 단계

npm, React 등과 관련된 자세한 예제를 확인하려면 IVS 브로드캐스트 SDK: 웹 안내서(실시간 스트리밍 가이드)를 참조하세요.

Android

보기 생성

먼저 자동 생성된 `activity_main.xml` 파일을 사용하여 앱의 간단한 레이아웃을 생성합니다. 레이아웃에는 토큰 추가를 위한 `EditText`, Join `Button`, 스테이지 상태 표시를 위한 `TextView`, 게시 전환을 위한 `CheckBox`가 포함되어 있습니다.



다음은 보기 뒤의 XML입니다.

```
<?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>
```

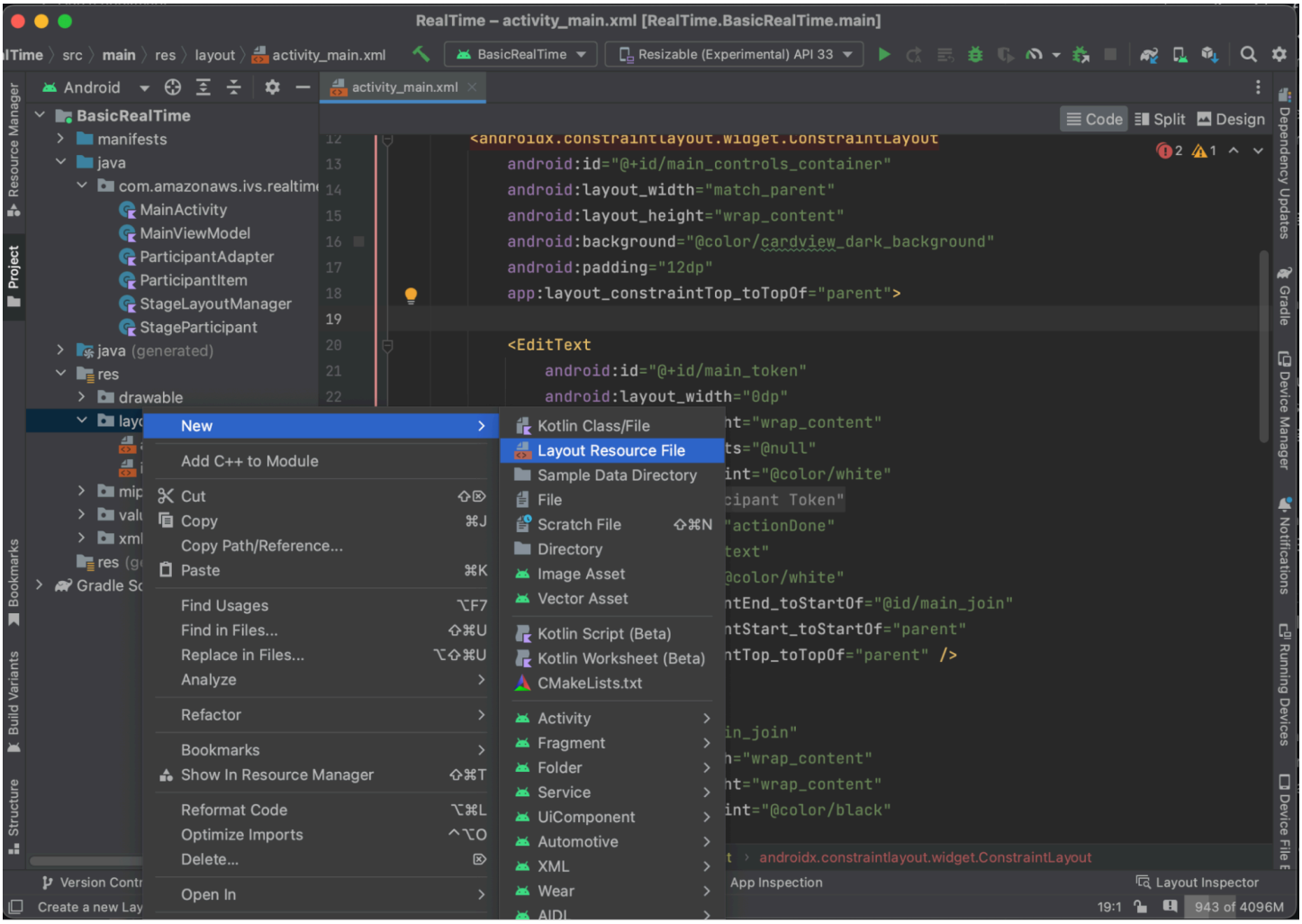
여기에서 몇 가지 문자열 ID를 참조했으므로 이제 전체 `strings.xml` 파일을 생성하겠습니다.

```
<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>
```

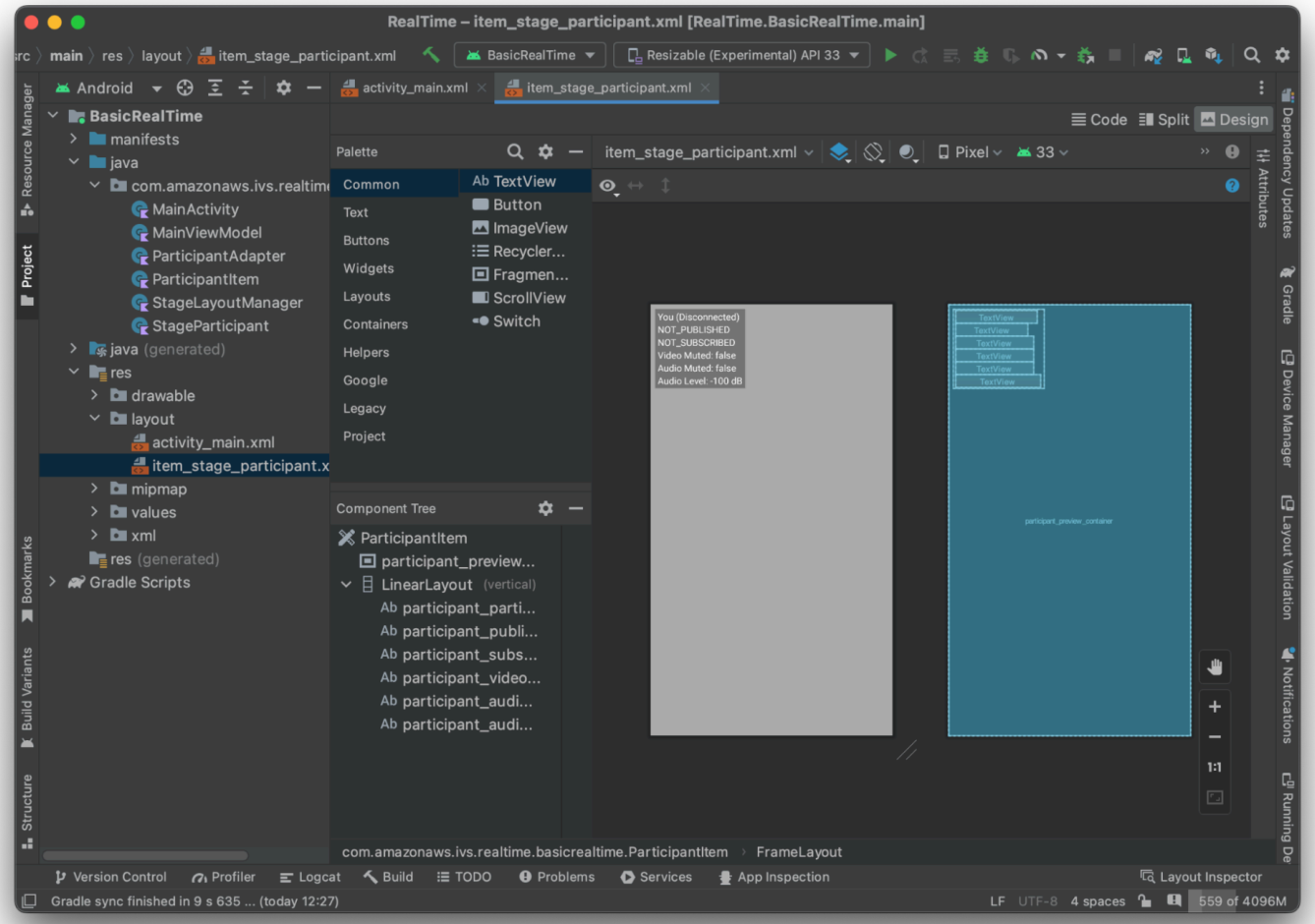
XML의 이러한 보기를 MainActivity.kt에 연결해 보겠습니다.

```
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)
}
```

이제 RecyclerView에 대한 항목 보기를 생성합니다. 이렇게 하려면 res/layout 디렉터리를 마우스 오른쪽 버튼으로 클릭하고 신규 > 레이아웃 리소스 파일을 선택합니다. 이 파일의 이름을 item_stage_participant.xml로 바꿉니다.



이 항목의 레이아웃은 간단합니다. 여기에는 참가자의 비디오 스트림을 렌더링하기 위한 보기와 참가자에 대한 정보를 표시하기 위한 레이블 목록이 포함되어 있습니다.



다음은 XML입니다.

```
<?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>
```

이 XML 파일은 아직 생성하지 않은 클래스인 `ParticipantItem`을 확장합니다. XML에는 전체 네임스페이스가 포함되어 있으므로 이 XML 파일을 네임스페이스로 업데이트해야 합니다. 이 클래스를 만들고 보기를 설정하되 지금은 비워 두겠습니다.

새 Kotlin 클래스 `ParticipantItem`을 생성합니다.

```
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)
    }
}
```

권한

카메라와 마이크를 사용하려면 사용자에게 권한을 요청해야 합니다. 이에 대한 표준 권한 흐름을 따릅니다.

```
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
    }
}
```

앱 상태

애플리케이션은 참가자를 로컬에서 `MainViewModel.kt` 추적하고 상태는 Kotlin을 `MainActivity` 사용하는 사용자에게 다시 전달됩니다. StateFlow

새 Kotlin 클래스 `MainViewModel`을 생성합니다.

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

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

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

}
```

`MainActivity.kt`에서 보기 모델을 관리합니다.

```
import androidx.activity.viewModels

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

`AndroidViewModel`과 이러한 Kotlin `ViewModel` 확장을 사용하려면 모듈의 `build.gradle` 파일에 다음을 추가해야 합니다.

```
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 어댑터

간단한 `RecyclerView.Adapter` 하위 클래스를 생성하여 참가자를 추적하고 스테이지 이벤트에서 `RecyclerView`를 업데이트합니다. 그러나 먼저 참가자를 나타내는 클래스가 필요합니다. 새 Kotlin 클래스 `StageParticipant`을 생성합니다.

```
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)
            }
        }
}
```

다음에 생성할 `ParticipantAdapter` 클래스에서 이 클래스를 사용하겠습니다. 먼저 클래스를 정의하고 참가자를 추적할 변수를 생성합니다.

```
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>()
```

또한 나머지 재정의를 구현하기 전에 `RecyclerView.ViewHolder`을 정의해야 합니다.

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

이를 사용하여 표준 `RecyclerView.Adapter` 재정의를 구현할 수 있습니다.

```
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)
    }
```

```
}
```

마지막으로, 참가자가 변경될 때 MainViewModel에서 직접적으로 호출할 새 메서드를 추가합니다. 이러한 메서드는 어댑터에 대한 표준 CRUD 작업입니다.

```
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])
    }
}
```

MainViewModel로 돌아가서 이 어댑터에 대한 참조를 생성하고 포함해야 합니다.

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

단계 상태

또한 MainViewModel 내에서 일부 스테이지 상태를 추적해야 합니다. 이제 이러한 속성을 정의해 보겠습니다.

```
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>()
```

스테이지에 참가하기 전에 미리 보기를 보려면 로컬 참가자를 즉시 생성합니다.

```
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)
}
```

ViewModel이 정리될 때 이러한 리소스를 정리해야 합니다. onCleared()를 즉시 재정의하므로 이러한 리소스를 정리하는 것을 잊지 않습니다.

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

이제 권한이 부여되는 즉시 로컬 streams 속성을 채우고 이전에 직접적으로 호출한 permissionsGranted 메서드를 구현합니다.

```
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)
    }
}
```

스테이지 SDK 구현

실시간 기능의 3가지 핵심 개념은 스테이지, 전략 및 렌더러입니다. 설계 목표는 작동하는 제품을 구축하는 데 필요한 클라이언트 측 로직의 수를 최소화하는 것입니다.

스테이지. 전략

우리의 `Stage.Strategy` 구현은 간단합니다.

```
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
}
```

요약하면, 내부 publishEnabled 상태를 기반으로 게시합니다. 게시하는 경우 이전에 수집한 스트림을 게시합니다. 마지막으로 이 샘플에서는 항상 다른 참가자를 구독하여 오디오와 비디오를 모두 수신합니다.

StageRenderer

StageRenderer 구현도 매우 간단하지만 함수 수를 감안할 때 훨씬 더 많은 코드가 포함되어 있습니다. 이 렌더러의 일반적인 접근 방식은 SDK가 참가자에 대한 변경 사항을 알릴 때 참가자 ParticipantAdapter를 업데이트하는 것입니다. 로컬 참가자가 참가하기 전에 카메라 미리 보기를 볼 수 있도록 직접 관리하기로 결정했기 때문에 로컬 참가자를 다르게 처리하는 특정 시나리오가 있습니다.

```
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) {}
}
```

커스텀 구현 RecyclerView LayoutManager

다른 수의 참가자를 배치하는 것은 복잡할 수 있습니다. 참가자가 전체 상위 보기의 프레임을 차지하도록 하되 각 참가자 구성을 독립적으로 처리하지 않으려고 합니다. 이를 쉽게 수행할 수 있도록 RecyclerView.LayoutManager을 구현하는 과정을 살펴보겠습니다.

GridLayoutManager를 확장해야 하는 또 다른 새 클래스인 StageLayoutManager를 생성합니다. 이 클래스는 흐름 기반 행/열 레이아웃의 참가자 수를 기준으로 각 참가자의 레이아웃을 계산하도록 설계되었습니다. 각 행은 다른 행과 높이가 같지만 열은 행마다 너비가 다를 수 있습니다. 이 동작을 사용자 정의하는 방법에 대한 설명은 layouts 변수 위의 코드 주석을 참조하세요.

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

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

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

    companion object {
        /**
         * This 2D array contains the description of how the grid of participants
 should be rendered
         * The index of the 1st dimension is the number of participants needed to
 active that configuration
         * Meaning if there is 1 participant, index 0 will be used. If there are 5
 participants, index 4 will be used.
         *
         * The 2nd dimension is a description of the layout. The length of the array is
 the number of rows that
         * will exist, and then each number within that array is the number of columns
 in each row.
         *
         * See the code comments next to each index for concrete examples.
         *
         * This can be customized to fit any layout configuration needed.
         */
        val layouts: List<List<Int>> = listOf(
            // 1 participant
            listOf(1), // 1 row, full width
            // 2 participants
            listOf(1, 1), // 2 rows, all columns are full width
            // 3 participants
```

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

    init {
        spanSizeLookup = object : SpanSizeLookup() {
            override fun getSpanSize(position: Int): Int {
                if (itemCount <= 0) {
                    return 1
                }
                // Calculate the row we're in
                val config = layouts[itemCount - 1]
                var row = 0
                var curPosition = position
                while (curPosition - config[row] >= 0) {
                    curPosition -= config[row]
                    row++
                }
                // spanCount == max spans, config[row] = number of columns we want
                // So spanCount / config[row] would be something like 6 / 3 if we want
3 columns.
                // So this will take up 2 spans, with a max of 6 is 1/3rd of the view.
                return spanCount / config[row]
```

```
            }
        }
    }

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

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

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

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

MainActivity.kt로 돌아가서 RecyclerView에 대한 어댑터 및 레이아웃 관리자를 설정해야 합니다.

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

UI 작업 연결

거의 다 되었습니다. 몇 가지 UI 작업만 연결하면 됩니다.

먼저 MainActivity가 MainViewModel의 StateFlow 변경 사항을 관찰하도록 합니다.

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

다음으로 Join 버튼과 Publish 확인란에 리스너를 추가합니다.

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

지금 구현하는 MainViewModel의 위 직접 호출 기능은 모두 다음과 같습니다.

```
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
}
```

참가자 렌더링

마지막으로 SDK에서 수신하는 데이터를 이전에 생성한 참가자 항목에 렌더링해야 합니다. `RecyclerView` 로직이 이미 완성되었으므로 `ParticipantItem`에서 `bind` API를 구현하기만 하면 됩니다.

먼저 empty 함수를 추가한 다음 단계별로 살펴보겠습니다.

```
fun bind(participant: StageParticipant) {

}
```

먼저 쉬움 상태, 참가자 ID, 게시 상태 및 구독 상태를 처리하겠습니다. 이를 위해 `TextViews`를 직접 업데이트합니다.

```
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
```

다음으로 오디오 및 비디오 음소거 상태를 업데이트하겠습니다. 음소거 상태를 얻으려면 streams 배열에서 `ImageDevice`와 `AudioDevice`를 찾아야 합니다. 성능을 최적화하기 위해 마지막으로 연결된 디바이스 ID를 기억합니다.

```
// 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"
}
```

마지막으로 imageDevice에 대한 미리 보기를 렌더링하려고 합니다.

```
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
```

그리고 audioDevice의 오디오 통계를 표시합니다.

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

```
    }
}
audioDeviceUrn = newAudioStream?.device?.descriptor?.urn
```

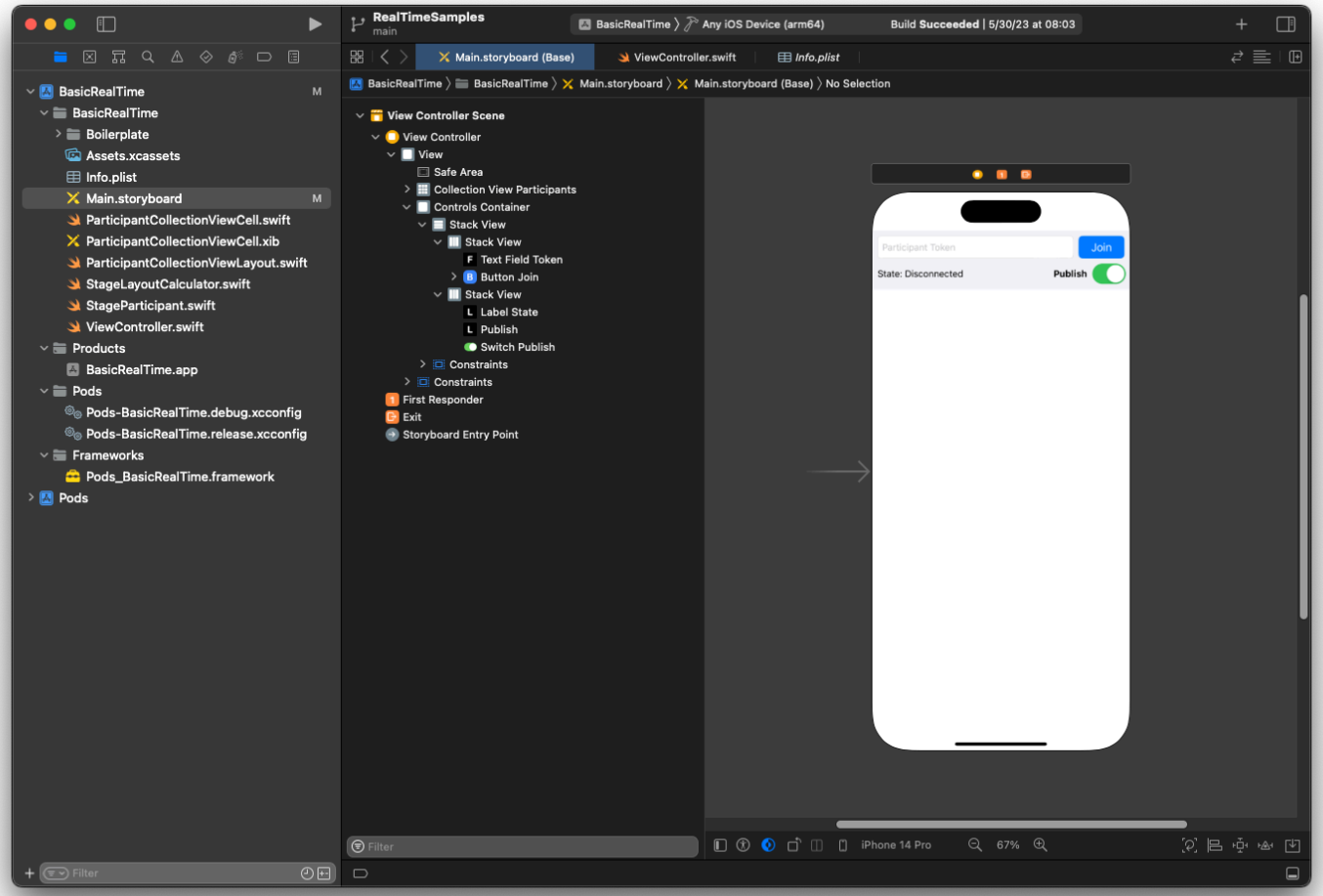
iOS

보기 생성

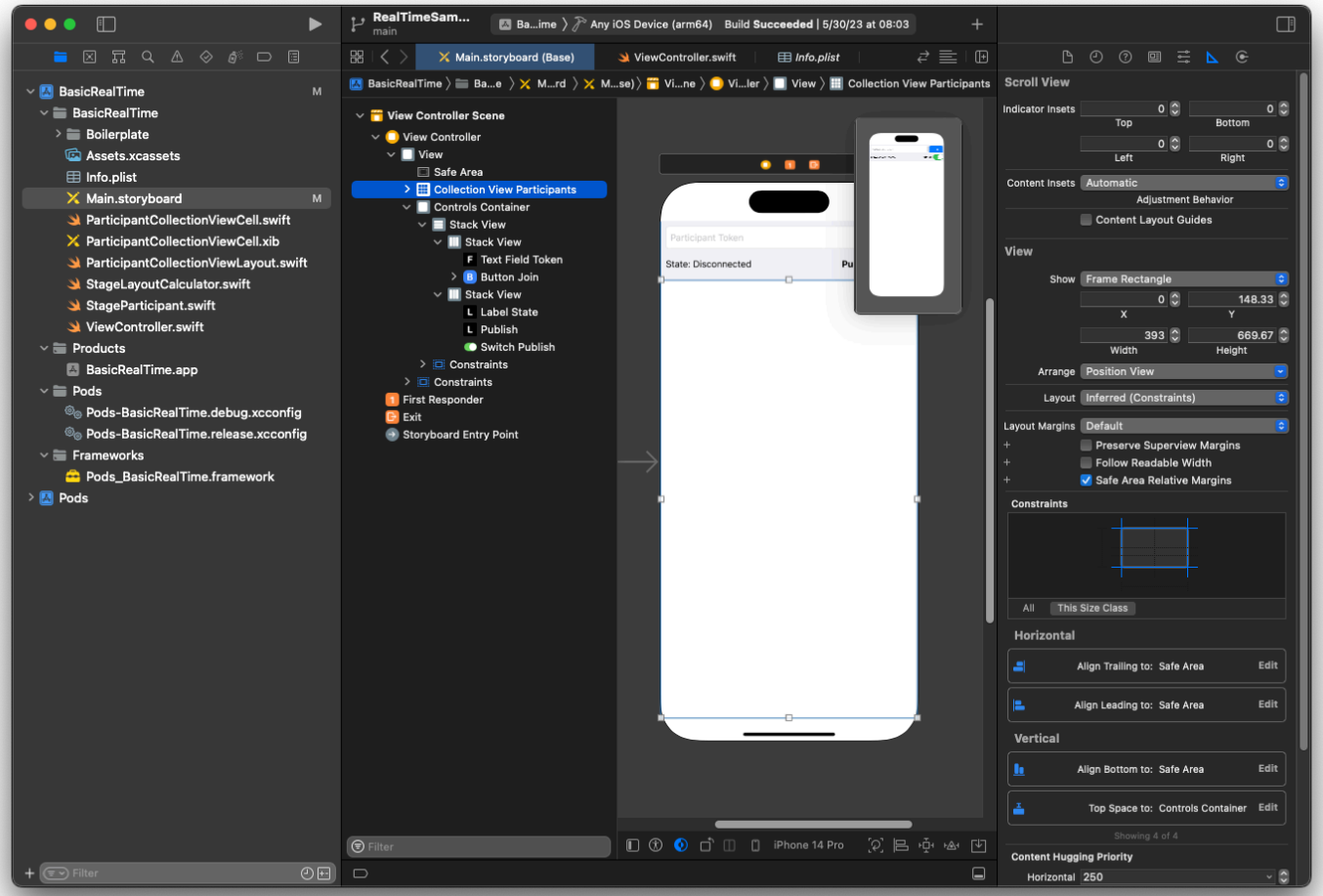
먼저 자동 생성된 `ViewController.swift` 파일을 사용하여 `AmazonIVSBroadcast`를 가져온 다음 링크에 `@IBOutlets`를 몇 개 추가합니다.

```
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!
```

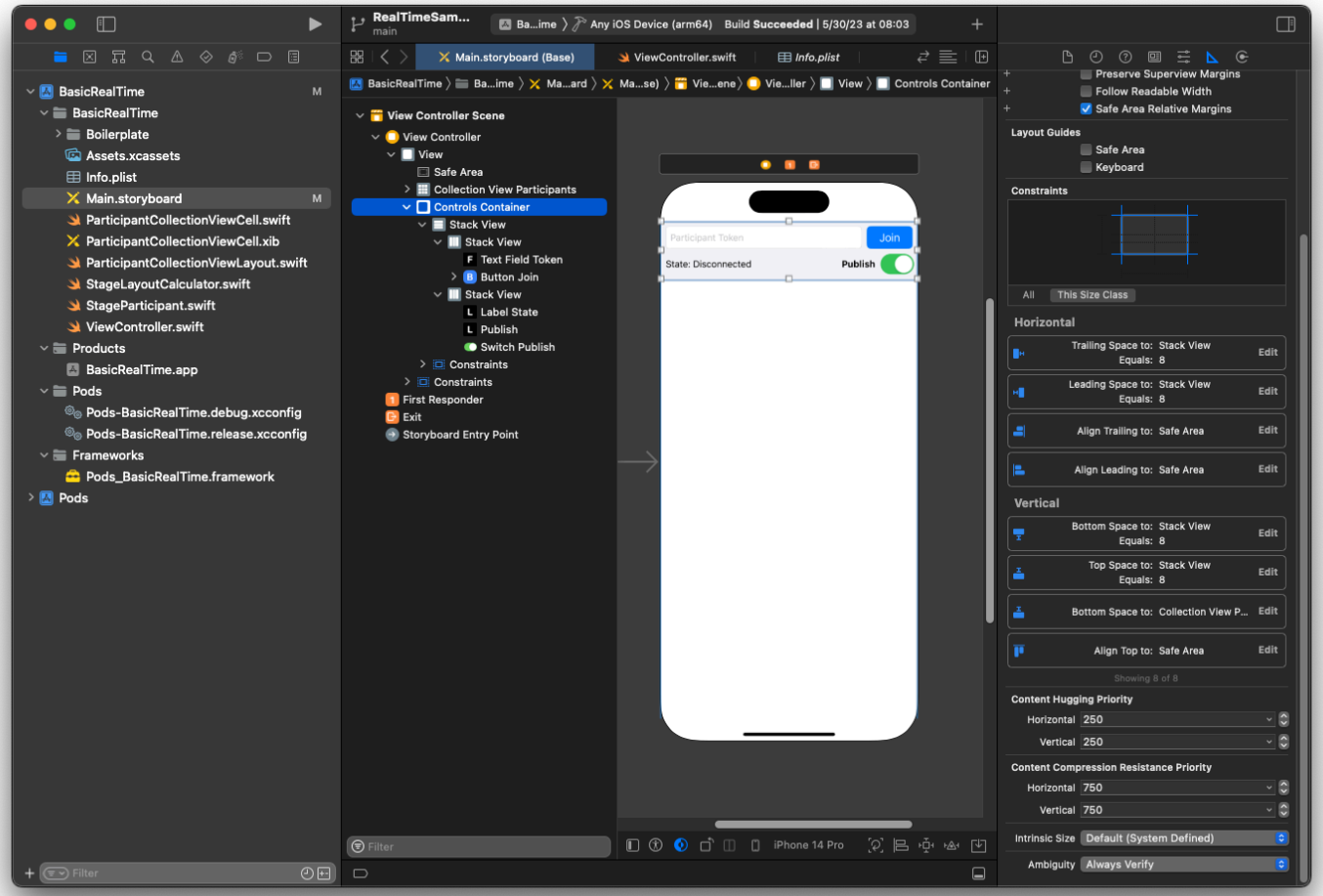
이제 이러한 보기를 생성하고 `Main.storyboard`에서 연결합니다. 사용할 보기 구조는 다음과 같습니다.



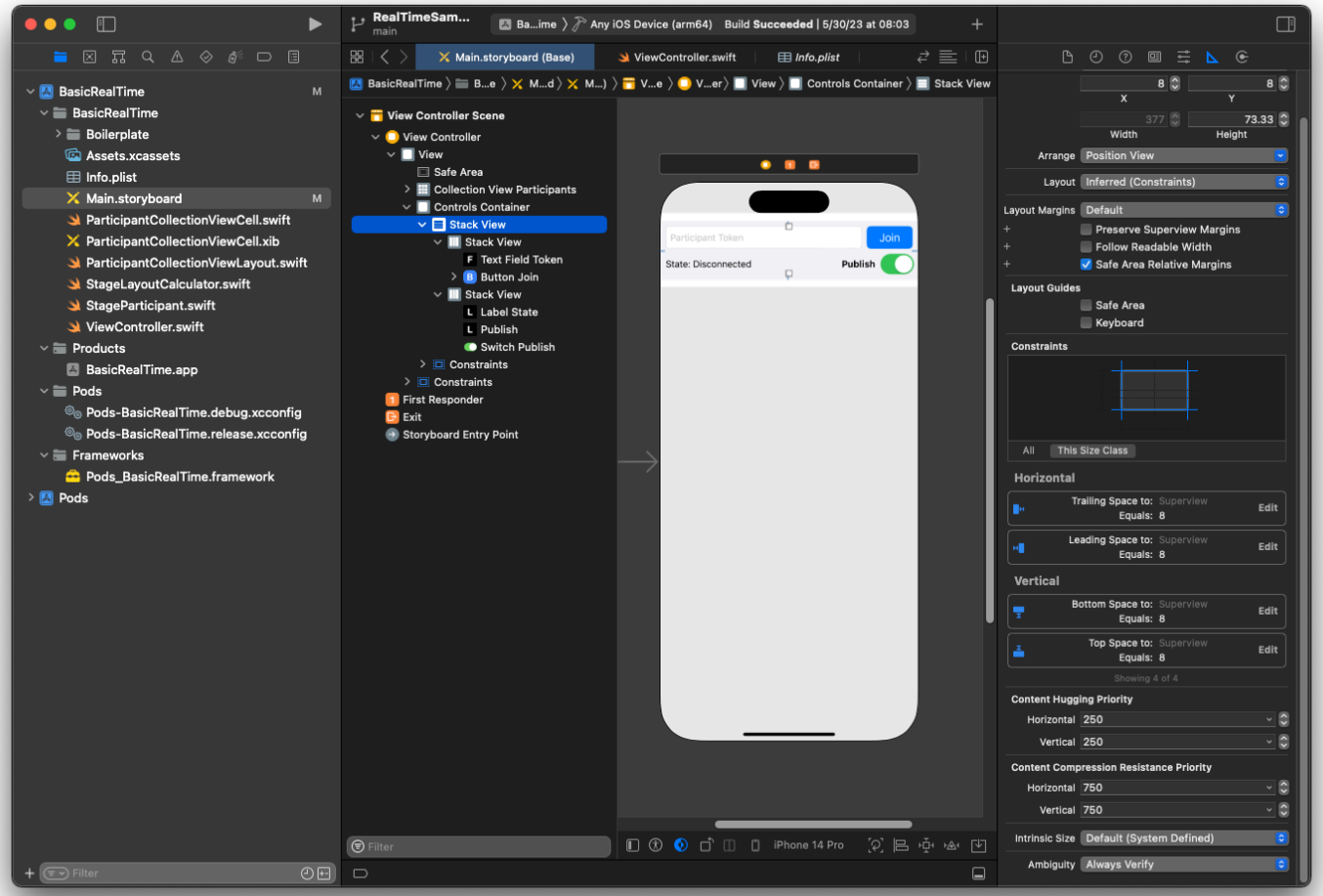
AutoLayout 구성을 위해서는 세 가지 보기를 사용자 지정해야 합니다. 첫 번째 보기는 컬렉션 보기 참가자(`UICollectionView`)입니다. 선행, 후행 및 하단을 안전한 영역에 바인딩합니다. 또한 상단을 컨트롤 컨테이너에 바인딩합니다.



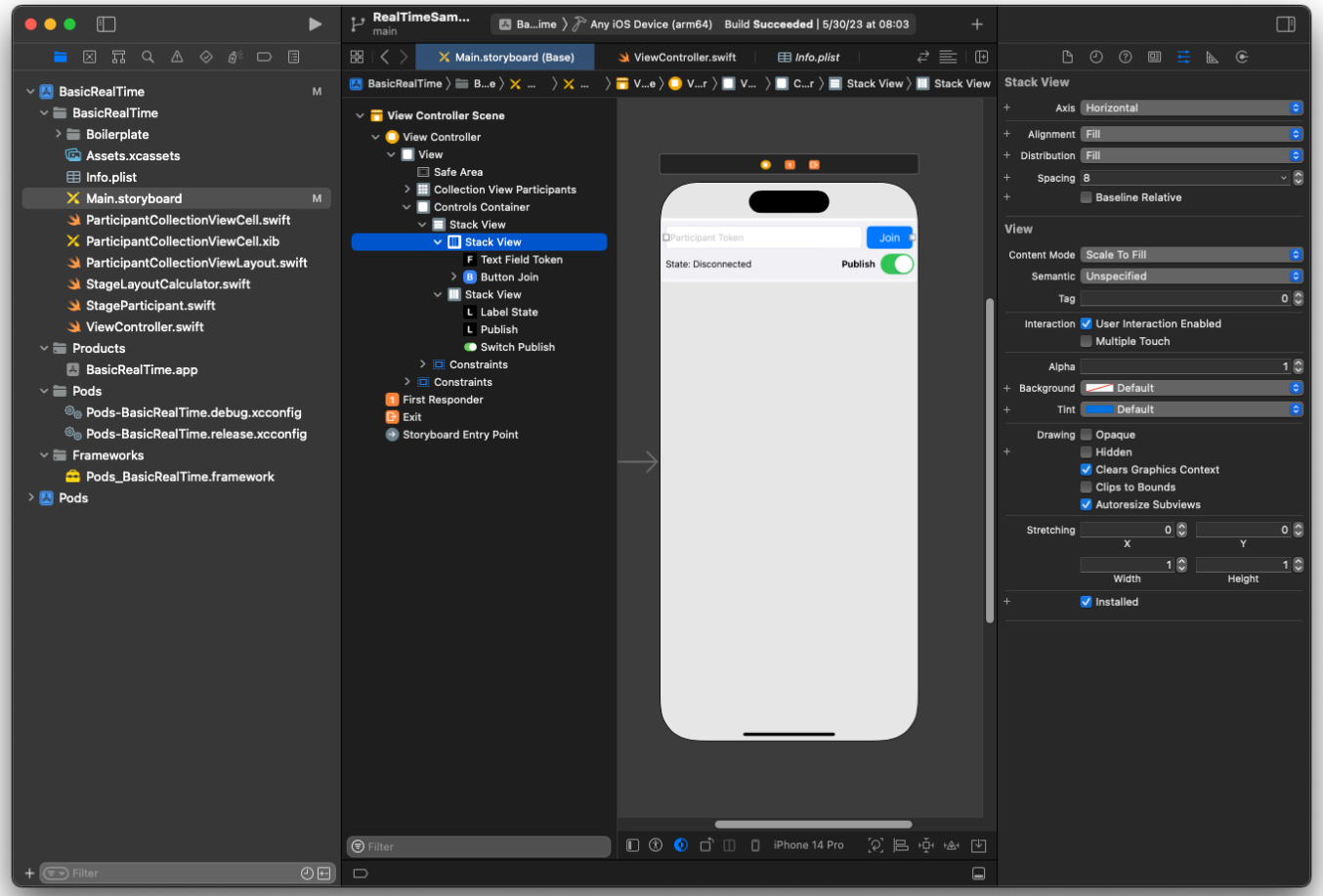
두 번째 보기는 컨트롤 컨테이너입니다. 선행, 후행 및 상단을 안전한 영역에 바인딩합니다.



세 번째이자 마지막 보기는 수직 스택 보기입니다. 상단, 선행, 후행 및 하단을 슈퍼뷰에 바인딩합니다. 스타일을 지정하려면 간격을 0 대신 8로 설정합니다.



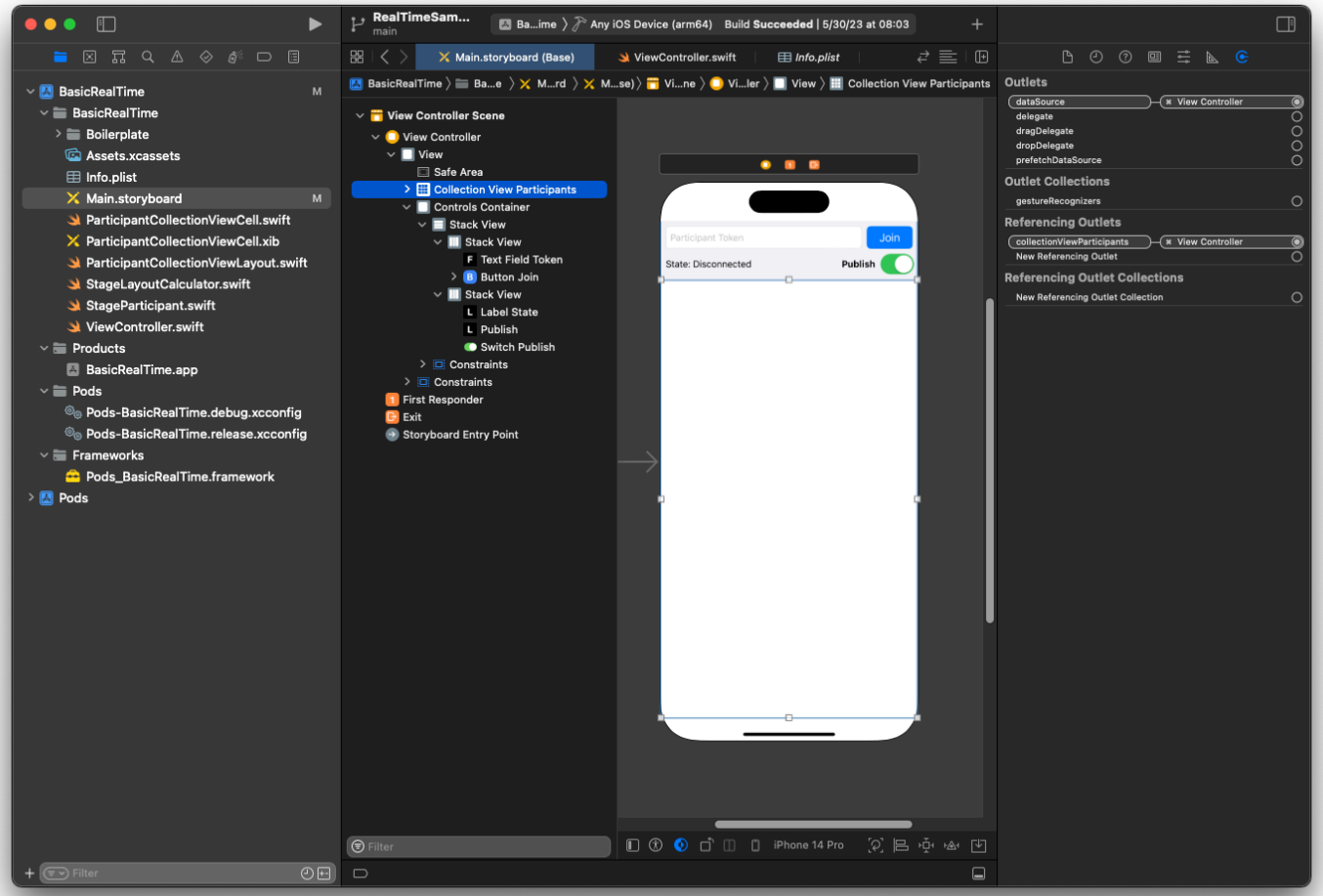
StackViewsUI는 나머지 뷰의 레이아웃을 처리합니다. 세 UI StackViews 모두에 대해 Fill을 정렬 및 배포로 사용합니다.



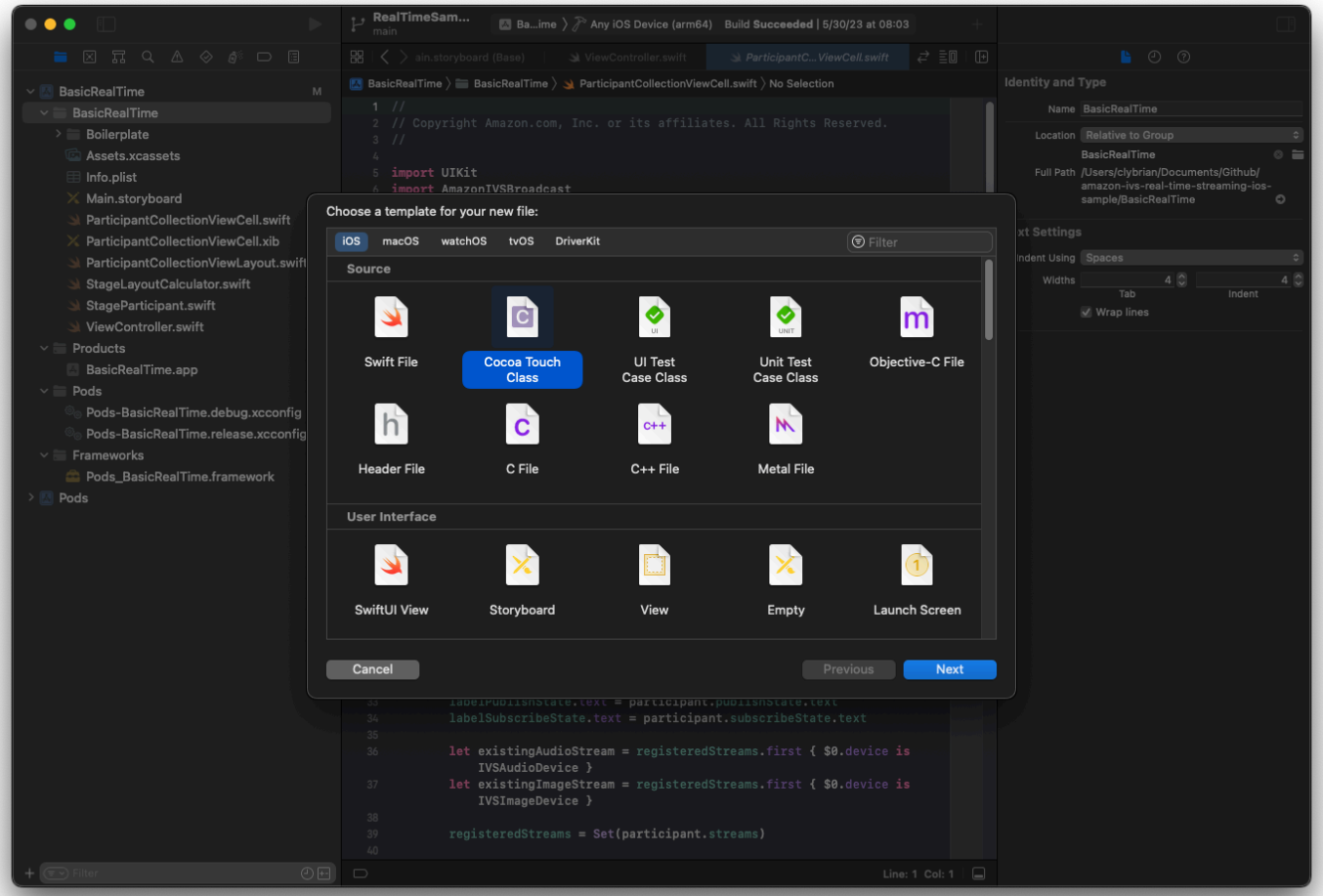
마지막으로 이들 보기를 `ViewController`에 연결하겠습니다. 위에서 다음 보기를 매핑합니다.

- 텍스트 필드 조인은 `textFieldToken`에 바인딩됩니다.
- 버튼 조인은 `buttonJoin`에 바인딩됩니다.
- 레이블 상태는 `labelState`에 바인딩됩니다.
- 스위치 게시는 `switchPublish`에 바인딩됩니다.
- 컬렉션 보기 참가자는 `collectionViewParticipants`에 바인딩됩니다.

또한 이 시간을 사용하여 컬렉션 보기 참가자 항목의 `dataSource`를 소유하는 `ViewController`로 설정합니다.



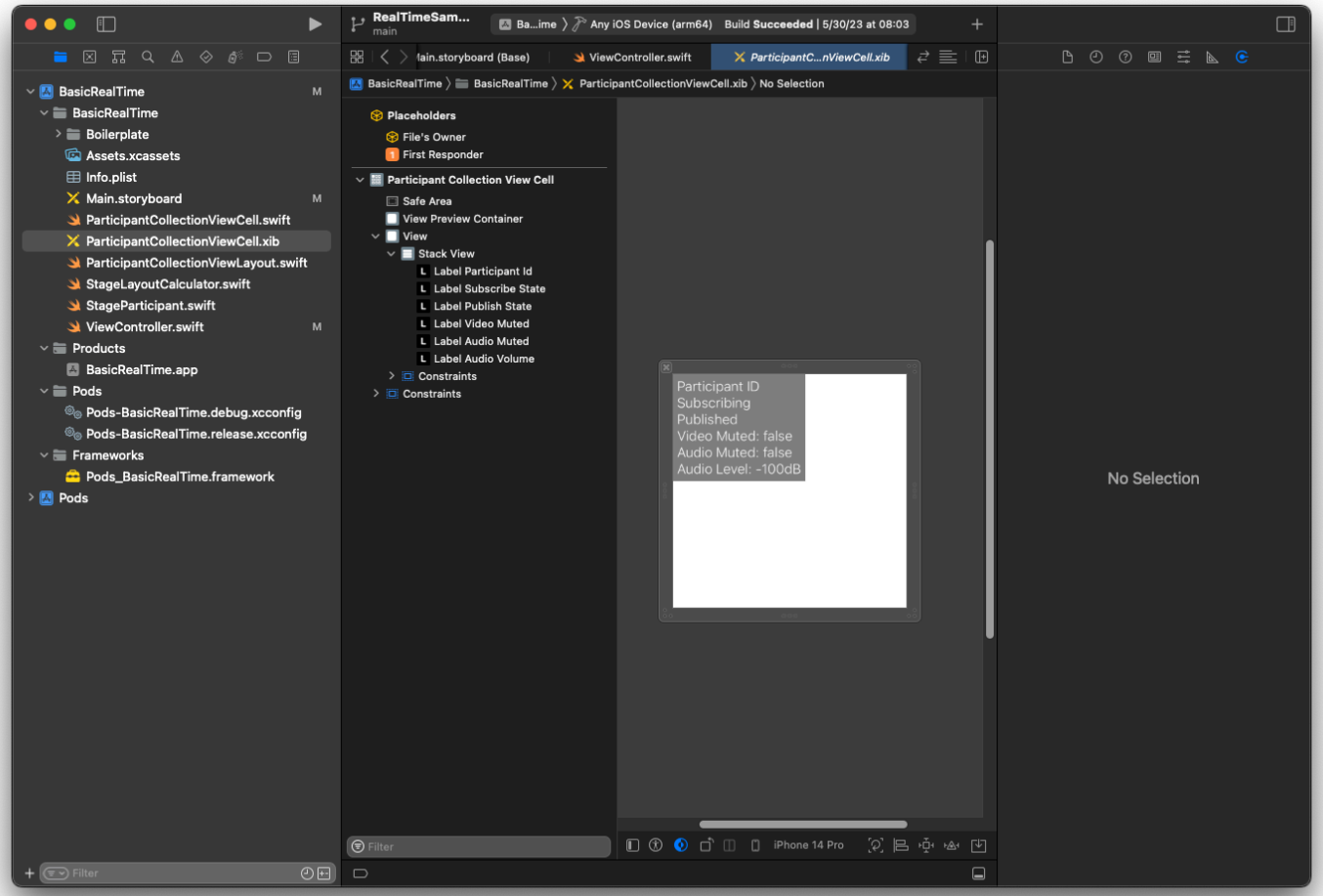
이제 참가자를 렌더링할 `UICollectionViewCell` 하위 클래스를 생성합니다. 먼저 새 Cocoa Touch Class 파일을 생성합니다.



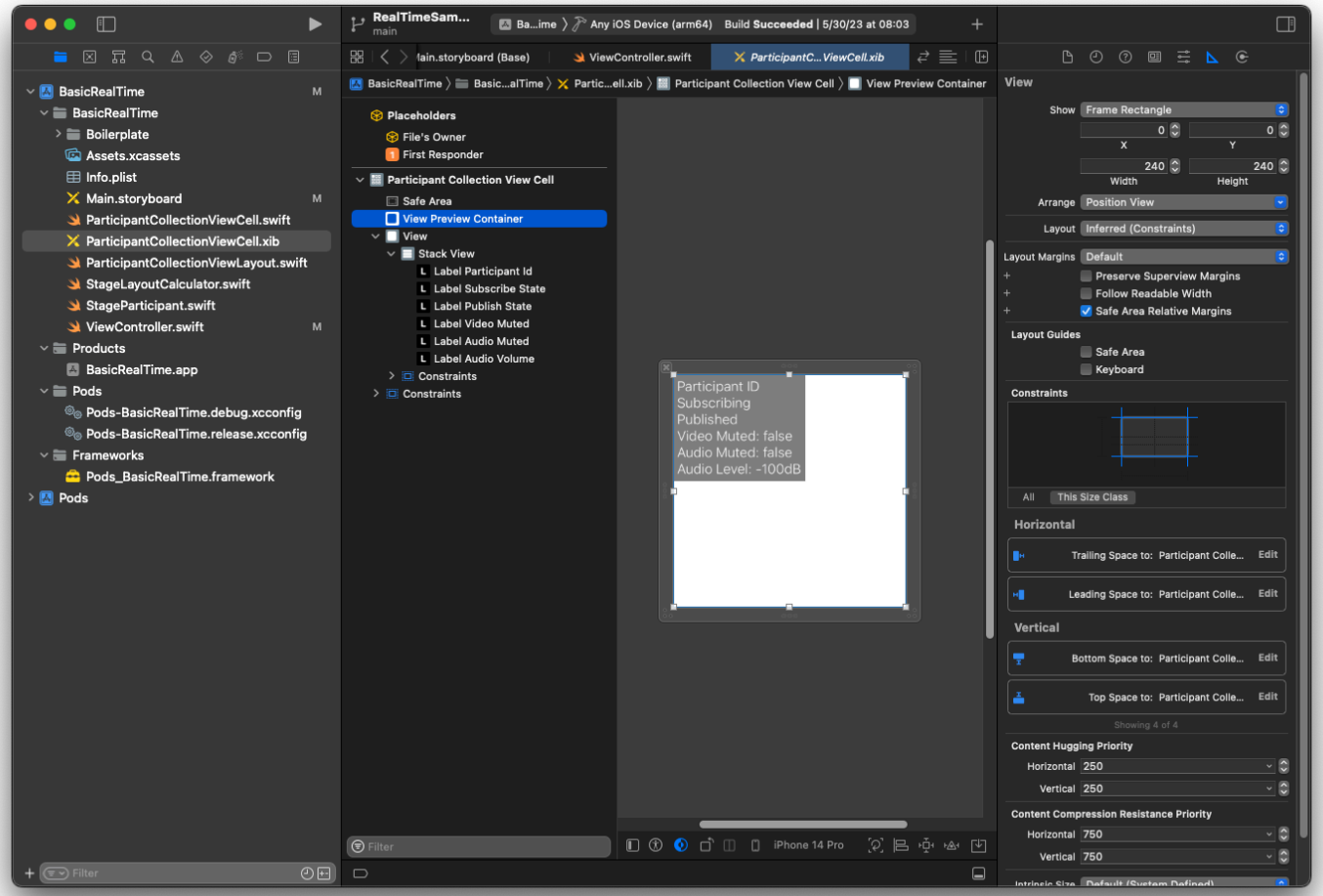
이름을 `ParticipantUICollectionViewCell`로 지정하고 Swift에서 `UICollectionViewCell`의 하위 클래스로 만듭니다. Swift 파일에서 다시 시작하여 연결할 `@IBOutlets`를 생성합니다.

```
import AmazonIVSBroadcast

class ParticipantCollectionViewCell: UICollectionViewCell {

    @IBOutlet private var viewPreviewContainer: UIView!
    @IBOutlet private var labelParticipantId: UILabel!
    @IBOutlet private var labelSubscribeState: UILabel!
    @IBOutlet private var labelPublishState: UILabel!
    @IBOutlet private var labelVideoMuted: UILabel!
    @IBOutlet private var labelAudioMuted: UILabel!
    @IBOutlet private var labelAudioVolume: UILabel!
```

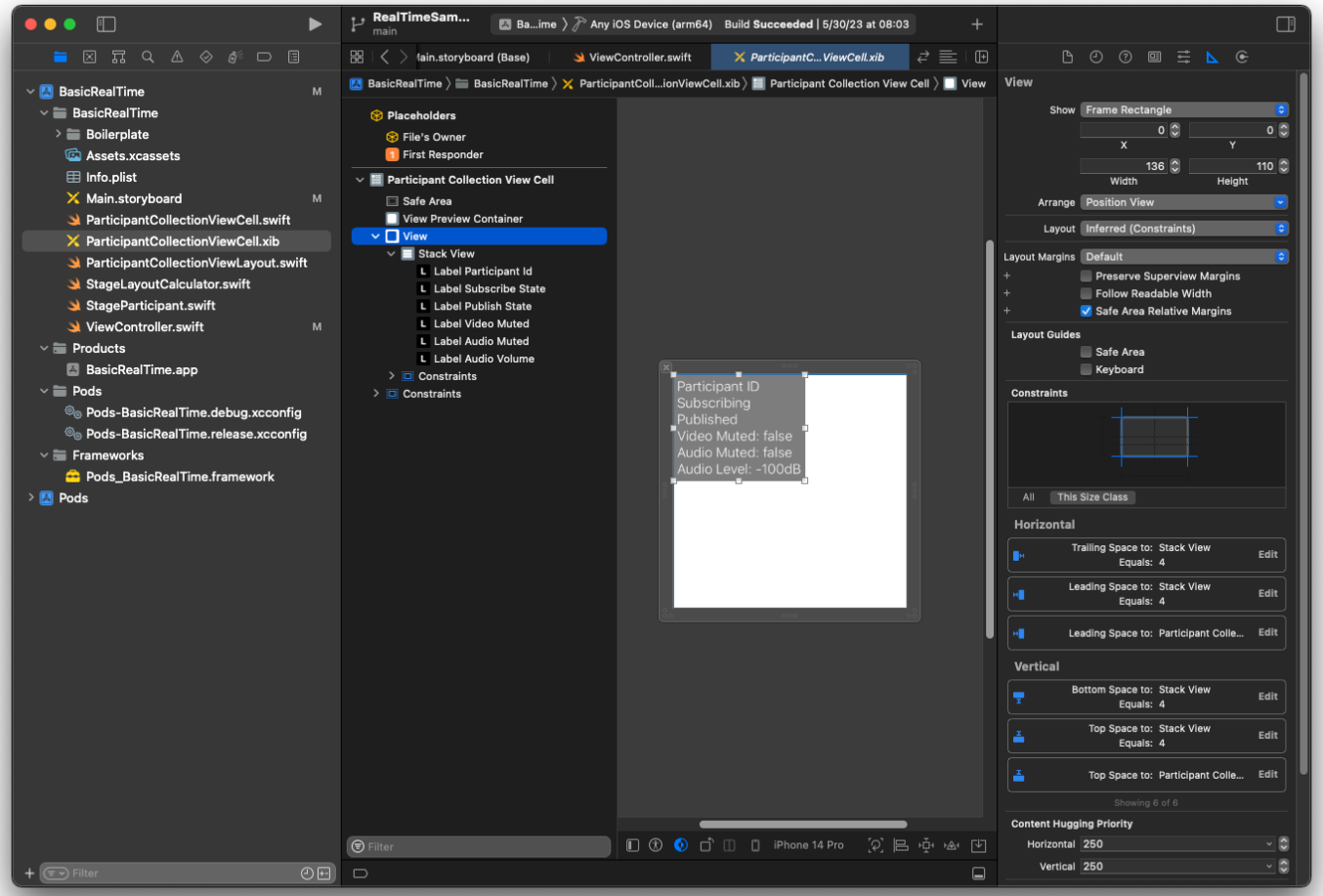
연결된 XIB 파일에서 다음과 같은 보기 계층을 생성합니다.



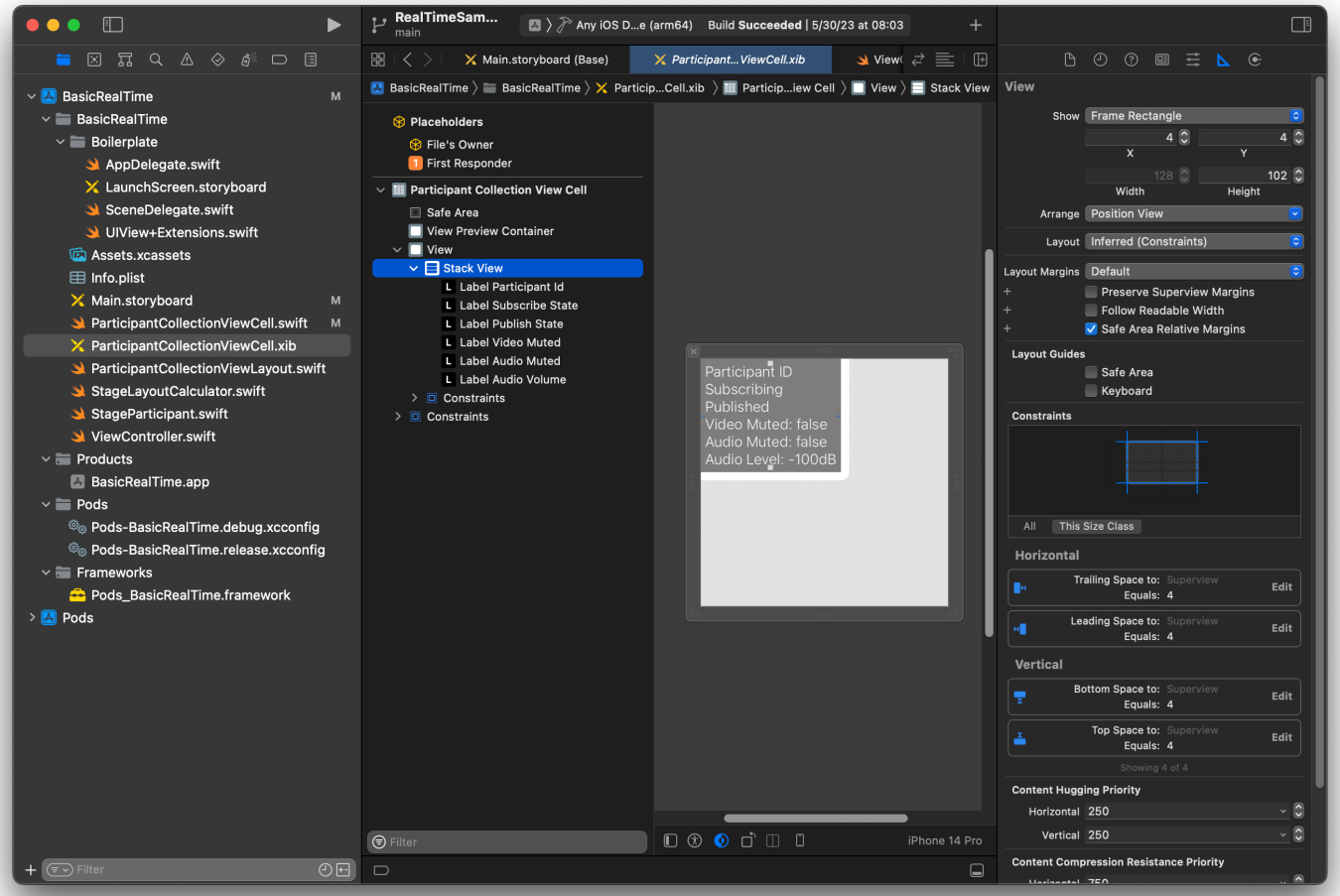
AutoLayout때문에 세 개의 뷰를 다시 수정하겠습니다. 첫 번째 보기는 보기 미리 보기 컨테이너입니다. 후행, 선행, 상단 및 하단을 참가자 컬렉션 보기 셀로 설정합니다.



두 번째 보기는 보기입니다. 선행과 상단을 참가자 컬렉션 보기 셀로 설정하고 값을 4로 변경합니다.



세 번째 보기는 Stack View입니다. 후행, 선행, 상단 및 하단을 슈퍼뷰로 설정하고 값을 4로 변경합니다.



권한 및 유휴 타이머

`ViewController`로 돌아가서 애플리케이션이 사용되는 동안 디바이스가 절전 모드로 전환되지 않도록 시스템 유휴 타이머를 비활성화하겠습니다.

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

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

다음으로 시스템에서 카메라 및 마이크 권한을 요청합니다.

```
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)
    }
}
```

앱 상태

이전에 생성한 레이아웃 파일을 사용하여 `collectionViewParticipants`를 구성해야 합니다.

```
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")
}
```

각 참가자를 나타내기 위해 StageParticipant라는 간단한 구조체를 생성합니다. 이를 ViewController.swift 파일에 포함하거나 새 파일을 생성할 수 있습니다.

```
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
    }
}
```

이러한 참가자를 추적하기 위해 ViewController에 참가자 배열을 프라이빗 속성으로 유지합니다.

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

이 속성은 이전에 스토리보드에서 연결된 UICollectionViewDataSource를 구동하는 데 사용됩니다.

```
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'")
        }
    }

}
```

스테이지에 참가하기 전에 미리 보기를 보려면 로컬 참가자를 즉시 생성합니다.

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

그 결과, 앱이 실행되는 즉시 참가자 셀이 렌더링되어 로컬 참가자를 나타냅니다.

사용자는 스테이지에 참가하기 전에 자신을 볼 수 있기를 원하므로 이전에 권한 처리 코드에서 직접적으로 호출되는 `setupLocalUser()` 메서드를 구현합니다. 카메라 및 마이크 참조를 `IVSLocalStageStream` 객체로 저장합니다.

```
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)
}
```

여기서는 SDK를 통해 디바이스의 카메라와 마이크를 찾아 로컬 streams 객체에 저장한 다음, 첫 번째 참가자(이전에 만든 로컬 참가자)의 streams 배열을 streams에 할당했습니다. 마지막으로 index가 0이고 changeType이 updated인 participantsChanged를 직접적으로 호출합니다. 이 함수는 멋진 애니메이션으로 UICollectionView를 업데이트하기 위한 도우미 함수입니다. 다음과 같습니다.

```
private func participantsChanged(index: Int, changeType: ChangeType) {
    switch changeType {
    case .joined:
        collectionViewParticipants?.insertItems(at: [IndexPath(item: index, section:
 0)])
    case .updated:
        // Instead of doing reloadItems, just grab the cell and update it ourselves. It
 saves a create/destroy of a cell
        // and more importantly fixes some UI flicker. We disable scrolling so the
 index path per cell
        // never changes.
        if let cell = collectionViewParticipants?.cellForItem(at: IndexPath(item:
 index, section: 0)) as? ParticipantCollectionViewCell {
            cell.set(participant: participants[index])
        }
    case .left:
        collectionViewParticipants?.deleteItems(at: [IndexPath(item: index, section:
 0)])
    }
}
```

아직 cell.set에 대해 걱정하지 마세요. 나중에 다루겠지만 여기서 참가자를 기반으로 셀의 내용을 렌더링할 것입니다.

ChangeType은 간단한 열거형입니다.

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

마지막으로 스테이지가 연결되어 있는지 여부를 추적하려고 합니다. 간단한 bool을 사용하여 자체적으로 업데이트될 때 무엇이 UI를 자동으로 업데이트하는지 추적합니다.

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

스테이지 SDK 구현

실시간 기능의 3가지 핵심 개념은 스테이지, 전략 및 렌더러입니다. 설계 목표는 작동하는 제품을 구축하는 데 필요한 클라이언트 측 로직의 수를 최소화하는 것입니다.

IVS StageStrategy

우리의 `IVSStageStrategy` 구현은 간단합니다.

```
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
    }
}
```

요약하자면, 게시 스위치가 '켜기' 위치에 있는 경우에만 게시하고, 게시하는 경우 이전에 수집한 스트림을 게시합니다. 마지막으로 이 샘플에서는 항상 다른 참가자를 구독하여 오디오와 비디오를 모두 수신합니다.

IVS StageRenderer

IVSStageRenderer 구현도 매우 간단하지만 함수 수를 감안할 때 훨씬 더 많은 코드가 포함되어 있습니다. 이 렌더러의 일반적인 접근 방식은 SDK가 참가자에 대한 변경 사항을 알릴 때 참가자 participants 배열을 업데이트하는 것입니다. 로컬 참가자가 참가하기 전에 카메라 미리 보기를 볼 수 있도록 직접 관리하기로 결정했기 때문에 로컬 참가자를 다르게 처리하는 특정 시나리오가 있습니다.

```
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)
    }
}
```

이 코드는 확장을 사용하여 연결 상태를 사용자에게 친숙한 텍스트로 변환합니다.

```
extension IVSStageConnectionState {
```

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

사용자 지정 UI 구현 CollectionViewLayout

다른 수의 참가자를 배치하는 것은 복잡할 수 있습니다. 참가자가 전체 상위 보기의 프레임을 차지하도록 하되 각 참가자 구성을 독립적으로 처리하지 않으려고 합니다. 이를 쉽게 수행할 수 있도록 UICollectionViewLayout을 구현하는 과정을 살펴보겠습니다.

UICollectionViewLayout을 확장해야 하는 또 다른 새 파일인 ParticipantCollectionViewLayout.swift를 생성합니다. 이 클래스는 곧 다룰 StageLayoutCalculator라는 다른 클래스를 사용합니다. 이 클래스는 각 참여자에 대해 계산된 프레임 값을 받은 다음 필요한 UICollectionViewLayoutAttributes 객체를 생성합니다.

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

더 중요한 것은 StageLayoutCalculator.swift 클래스입니다. 이 클래스는 흐름 기반 행/열 레이아웃의 참가자 수를 기준으로 각 참가자의 프레임을 계산하도록 설계되었습니다. 각 행은 다른 행과 높이가 같지만 열은 행마다 너비가 다를 수 있습니다. 이 동작을 사용자 정의하는 방법에 대한 설명은 layouts 변수 위의 코드 주석을 참조하세요.

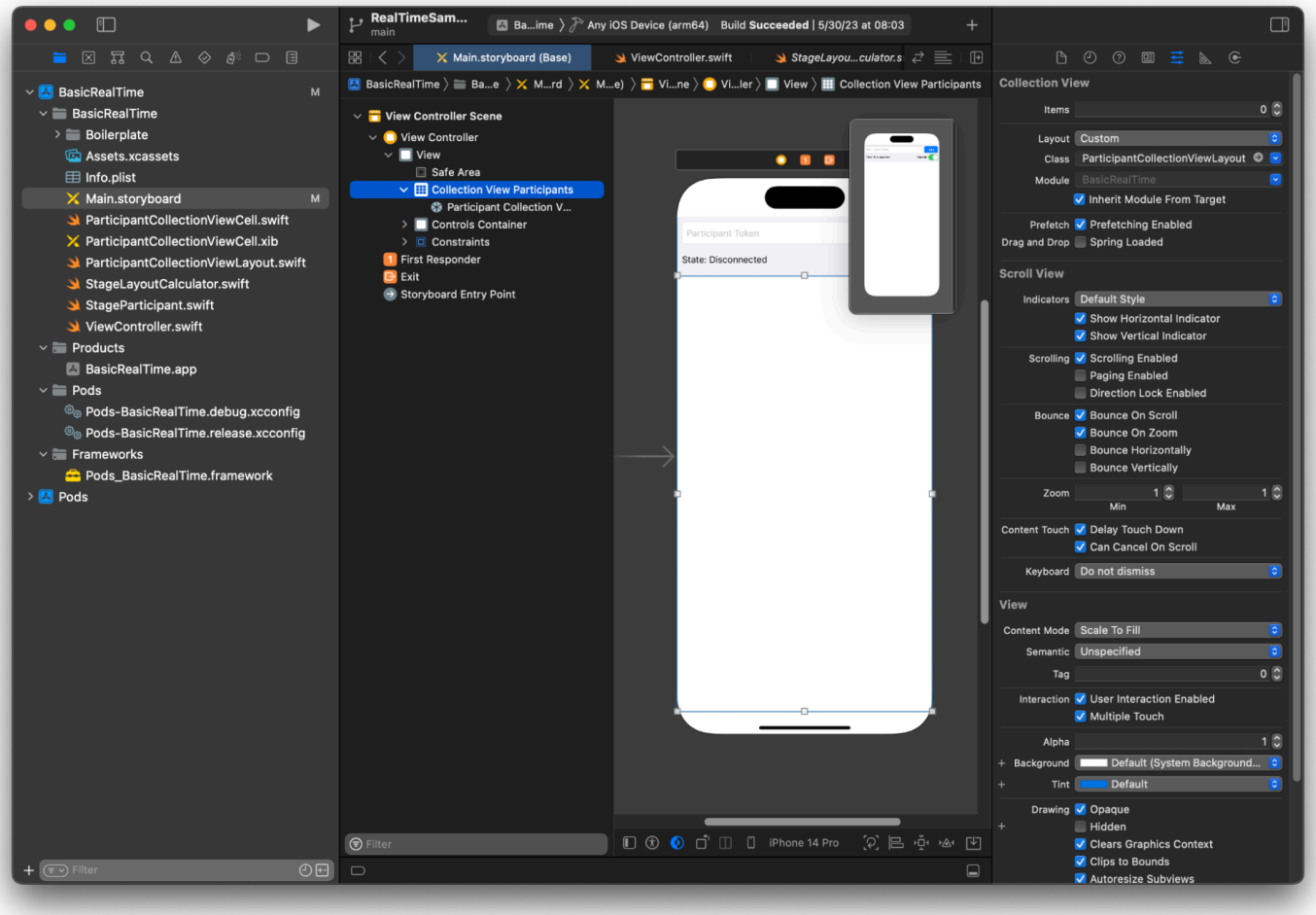
```
import Foundation
import UIKit

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
    }

}
```

`Main.storyboard`로 돌아가서 `UICollectionView`의 레이아웃 클래스를 방금 생성한 클래스로 설정해야 합니다.



UI 작업 연결

거의 다 되었습니다. 몇 가지 IBActions만 생성하면 됩니다.

먼저 Join 버튼을 처리하겠습니다. 이 버튼은 connectingOrConnected 값에 따라 다르게 응답합니다. 이미 연결되어 있으면 그냥 스테이지에서 나갑니다. 연결이 끊어지면 토큰 UITextField에서 텍스트를 읽고 해당 텍스트로 새 IVSStage를 생성합니다. 그런 다음 ViewController를 IVSStage에 대한 strategy, errorDelegate 및 renderer로 추가하고 마지막으로 스테이지에 비동기적으로 조인합니다.

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

연결해야 하는 또 다른 UI 작업은 게시 스위치입니다.

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

참가자 렌더링

마지막으로 SDK에서 수신하는 데이터를 이전에 생성한 참가자 셀에 렌더링해야 합니다. UICollectionView 로직이 이미 완성되었으므로 ParticipantCollectionViewCell.swift에서 set API를 구현하기만 하면 됩니다.

먼저 empty 함수를 추가한 다음 단계별로 살펴보겠습니다.

```
func set(participant: StageParticipant) {

}
```

먼저 쉬움 상태, 참가자 ID, 게시 상태 및 구독 상태를 처리하겠습니다. 이를 위해 UILabels를 직접 업데이트합니다.

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

게시 및 구독 열거형의 텍스트 속성은 로컬 확장에서 가져온 것입니다.

```
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()
        }
```

```
    }
}
```

다음으로 오디오 및 비디오 음소거 상태를 업데이트하겠습니다. 음소거 상태를 얻으려면 streams 배열에서 IVSImageDevice와 IVSAudioDevice를 찾아야 합니다. 성능을 최적화하기 위해 마지막으로 연결된 디바이스를 기억합니다.

```
// 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)"
```

마지막으로 imageDevice에 대한 미리 보기를 렌더링하고 audioDevice의 오디오 통계를 표시하려고 합니다.

```
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"
}
```

마지막으로 생성해야 하는 함수는 보기에 참가자의 미리 보기를 추가하는 updatePreview()입니다.

```
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)
        }
    }
}
```

위에서는 서브뷰를 더 쉽게 포함할 수 있도록 UIView에서 도우미 함수를 사용합니다.

```
extension UIView {
    func addSubviewMatchFrame(_ view: UIView) {
        view.translatesAutoresizingMaskIntoConstraints = false
        self.addSubview(view)
        NSLayoutConstraint.activate([
            view.topAnchor.constraint(equalTo: self.topAnchor, constant: 0),
            view.bottomAnchor.constraint(equalTo: self.bottomAnchor, constant: 0),
            view.leadingAnchor.constraint(equalTo: self.leadingAnchor, constant: 0),
            view.trailingAnchor.constraint(equalTo: self.trailingAnchor, constant: 0),
        ])
    }
}
```

Amazon IVS 실시간 스트리밍 모니터링

스테이지 세션이란 무엇인가요?

첫 번째 참가자가 스테이지에 참가하면 스테이지 세션이 시작되고 마지막 참가자가 스테이지에 게시를 중지하면 몇 분 후에 스테이지 세션이 종료됩니다. 스테이지 세션은 이벤트와 참가자를 단기 세션으로 분리하여 장기 세션 디버깅을 지원합니다.

스테이지 세션 및 참가자 보기

콘솔 지침

1. Amazon IVS 콘솔을 엽니다.

   (또는 AWS Management Console을 통해서 Amazon IVS 콘솔에 액세스할 수 있습니다
2. 탐색 창에서 스테이지를 선택합니다. (탐색 창이 축소된 경우 먼저 햄버거 아이콘을 선택하여 엽니다.)
3. 스테이지를 선택하여 해당 세부 정보 페이지로 이동합니다.
4. 스테이지 세션 섹션이 표시될 때까지 페이지를 아래로 스크롤한 다음 세부 정보 페이지를 볼 스테이지 세션을 선택합니다.
5. 세션의 참가자를 보려면 참가자 섹션이 표시될 때까지 아래로 스크롤한 다음에 Amazon CloudWatch 지표를 포함한 세부 정보 페이지를 볼 참가자를 선택합니다.

참가자에 대한 이벤트 보기

스테이지에 참가하거나 스테이지에 게시하는 동안 오류가 발생하는 등 스테이지에서 참가자의 상태가 변경될 때 이벤트가 전송됩니다. 모든 오류가 이벤트를 일으키는 것은 아닙니다. 예를 들어 클라이언트 측 네트워크 오류와 토큰 서명 오류는 이벤트로 전송되지 않습니다. 클라이언트 애플리케이션에서 이러한 오류를 처리하려면 IVS 브로드캐스트 SDK를 사용합니다.

콘솔 지침

1. 위의 지침에 따라 참가자 세부 정보 페이지로 이동합니다.

2. 이벤트 섹션이 표시될 때까지 아래로 스크롤합니다. 그러면 참가자 이벤트의 정렬된 목록이 표시됩니다. 참가자를 위해 방출되는 이벤트에 대한 자세한 내용은 Amazon IVS에서 Amazon EventBridge 사용을 참조하세요.

CLI 지침

AWS CLI를 사용하여 스테이지 세션 이벤트에 액세스하는 것은 고급 옵션이며, 먼저 머신에서 CLI를 다운로드하고 구성해야 합니다. 자세한 내용은 AWS Command Line Interface 사용 설명서를 참조하세요.

1. 스테이지 세션을 나열하여 스테이지 세션을 찾습니다.

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

2. 스테이지 세션에 대한 참가자를 나열하여 참가자를 찾습니다.

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

3. 스테이지 세션과 참가자에 대한 이벤트를 나열합니다.

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

다음은 `list-participant-events` 호출에 대한 샘플 응답입니다.

```
{
    "events": [
        {
            "eventTime": "2023-04-04T22:48:41+00:00",
            "name": "JOINED",
            "participantId": "AdRezBl021t0"
        },
        {
            "eventTime": "2023-04-04T22:48:41+00:00",
            "name": "SUBSCRIBE_STARTED",
            "participantId": "AdRezBl021t0",
            "remoteParticipantId": "Ou5b5n5XLMdC"
        },
        {
            "eventTime": "2023-04-04T22:49:45+00:00",
```

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

CloudWatch 지표 액세스

CloudWatch 지표를 사용하기 위해서는 웹 1.5.0 이상, Android 1.12.0 이상 또는 iOS 1.12.0 이상의 IVS 브로드캐스트 SDK 버전이 필요합니다.

CloudWatch 콘솔 지침

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. 측면 탐색에서 지표(Metrics) 드롭다운을 확장한 다음 모든 지표(All metrics)를 선택합니다.
3. 검색 탭에서 왼쪽에 있는 레이블이 없는 드롭다운을 사용하여 채널이 생성된 '홈' 리전을 선택합니다. 리전에 대한 자세한 내용은 글로벌 솔루션, 리전별 제어를 참조하세요. 지원되는 리전 목록은 AWS 일반 참조의 Amazon IVS 페이지를 참조하세요.
4. 검색 탭 하단 부분에서 IVSRealTime 네임스페이스를 선택하세요.
5. 다음 중 하나를 수행합니다.
   a. 검색 창에 리소스 ID(ARN의 일부, `arn:::ivs:stage/<resource id>`)를 입력합니다.

      그리고 나서 IVSRealTime > 스테이지 지표를 선택하세요.

   b. IVSRealTime이 AWS 네임스페이스 아래에서 선택 가능한 서비스로 나타나는 경우 선택하세요. Amazon IVS 실시간 스트리밍을 사용하고 Amazon CloudWatch에 지표를 전송하는 경우 나열됩니다. (IVSRealTime이 목록에 없으면 Amazon IVS 지표가 없습니다.)

      그리고 나서 필요할 경우 차원 그룹화를 선택하세요. 아래의 CloudWatch 지표에 사용 가능한 차원이 나열되어 있습니다.
6. 지표를 선택하여 그래프에 추가합니다. 아래의 CloudWatch 지표에 사용 가능한 지표가 나열되어 있습니다.

또한 스트림 세션의 세부 정보 페이지에서 CloudWatch에서 보기(View in CloudWatch) 버튼을 선택하여 스트림 세션의 CloudWatch 차트에 액세스할 수 있습니다.

CLI 지침

AWS CLI를 사용하여 지표에 액세스할 수도 있습니다. 그러려면 먼저 시스템에 CLI를 다운로드하여 구성해야 합니다. 자세한 내용은 AWS 명령줄 인터페이스 사용 설명서를 참조하세요.

그런 다음, AWS CLI를 사용하여 Amazon 실시간 스트리밍 IVS 지표에 액세스하려면 다음을 수행합니다.

- 명령 프롬프트에서 다음을 실행합니다.

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

자세한 내용은 Amazon CloudWatch 사용 설명서에서 Amazon CloudWatch 지표 사용을 참조하세요.

CloudWatch 지표: IVS 실시간 스트리밍

Amazon IVS는 AWS/IVSRealTime 네임스페이스에서 다음 지표를 제공합니다.

CloudWatch 지표를 사용하기 위해서는 반드시 웹 브로드캐스트 SDK 1.5.2 이상을 사용해야 합니다.

차원의 유효한 값은 다음과 같을 수 있습니다.

- `Stage` 차원은 리소스 ID(ARN의 일부, `arn:::stage/<resource id>`)입니다.
- `Participant` 차원은 `participantID`입니다.
- `SimulcastLayer`는 "video" `MediaType`의 경우 "hi", "mid", "low" 또는 "no-rid"이고 "audio" `MediaType`의 경우 "disabled"입니다. 이 값도 비워둘 수 있습니다.
- `MediaType` 차원은 '비디오' 또는 '오디오'(문자열) 입니다.

지표	차원	Description
`DownloadPacketLoss`	`Stage`	각 샘플은 특정 구독자가 IVS 서버에서 다운로드하는 동안 손실한 패킷의 백분율을 나타냅니다. 단위: 백분율

지표	차원	Description
		유효 통계: 평균, 최대, 최소 - 구성된 간격 동안 패킷 손실의 평균 수, 가장 많은 수 또는 가장 적은 수(각각)입니다.
DownloadPacketLoss	Stage,Participant	게시자이기도 한 구독자의 경우 참가자별로 DownloadPacketLoss 를 필터링합니다. 샘플은 구독자가 IVS 서버에서 다운로드하는 동안 손실한 패킷의 백분율을 나타냅니다. 참가자가 게시자이기도 한 경우에만 샘플이 내보내집니다. 단위: 백분율 유효 통계: 평균, 최대, 최소 - 구성된 간격 동안 드롭된 프레임의 평균 수, 가장 많은 수 또는 가장 적은 수(각각)입니다.
DroppedFrames	Stage	각 샘플은 특정 구독자가 드롭한 프레임의 백분율을 나타냅니다. 단위: 백분율 유효 통계: 평균, 최대, 최소 - 구성된 간격 동안 드롭된 프레임의 평균 수, 가장 많은 수 또는 가장 적은 수(각각)입니다.
DroppedFrames	Stage,Participant	게시자이기도 한 구독자의 경우 참가자별로 DroppedFrames 를 필터링합니다. 샘플은 구독 참가자와 스테이지의 모든 게시자 사이에서 삭제된 프레임의 백분율을 나타냅니다. 참가자가 게시자이기도 한 경우에만 샘플이 내보내집니다. 단위: 백분율 유효 통계: 평균, 최대, 최소 - 구성된 간격 동안 드롭된 프레임의 평균 수, 가장 많은 수 또는 가장 적은 수(각각)입니다.

지표	차원	Description
PublishBitrate	Stage	방출된 샘플은 특정 게시자가 비디오와 오디오 데이터를 모두 전송하는 전체 속도(모든 동시 방송 계층 전반에서 집계)를 나타냅니다. 단위: 초당 비트 유효 통계: 평균, 최대, 최소 - 구성된 간격 동안 비트레이트의 평균 수, 가장 많은 수 또는 가장 적은 수(각각)입니다.
PublishBitrate	Stage, Participant, SimulcastLayer, MediaType	참가자, 동시 방송 계층 및 미디어 종류를 기준으로 PublishBitrate 를 필터링하세요. 동시 방송 계층 ID는 브로드캐스트 SDK에 의해 설정됩니다. 동시 방송이 비활성화된 경우 이 계층 ID는 '비활성화됨'으로 설정됩니다. 미디어 종류는 비디오 또는 오디오입니다. 단위: 초당 비트 유효 통계: 평균, 최대, 최소 - 구성된 간격 동안 비트레이트의 평균 수, 가장 많은 수 또는 가장 적은 수(각각)입니다.
Publishers	Stage	스테이지에 게시하는 참가자의 수입니다. 단위: 개 유효 통계: 평균, 최대, 최소
PublishResolution	Stage, Participant, SimulcastLayer, MediaType	프레임 너비 또는 높이 중 작은 값 전반의 픽셀 수입니다. 예를 들어 크기가 1920x1080인 가로 프레임의 경우 게시 해상도는 1080입니다. 크기가 720x1280인 세로 프레임의 경우 게시 해상도는 720입니다. 단위: 개 유효 통계: 평균, 최대, 최소

지표	차원	Description
SubscribeBitrate	Stage	방출된 샘플은 특정 구독자가 비디오와 오디오 데이터를 둘 다 수신하는 전체 속도를 나타냅니다. 단위: 초당 비트 유효 통계: 평균, 최대, 최소 - 구성된 간격 동안 비트레이트의 평균 수, 가장 많은 수 또는 가장 적은 수(각각)입니다.
SubscribeBitrate	Stage,Participant,MediaType	게시자이기도 한 구독자의 경우 참가자별로 SubscribeBitrate 를 필터링합니다. 샘플은 특정 구독자가 특정 MediaType 을 수신하는 비트레이트를 나타냅니다. 구독 참가자가 게시하는 동안에만 샘플이 내보내집니다. 단위: 초당 비트 유효 통계: 평균, 최대, 최소 - 구성된 간격 동안 비트레이트의 평균 수, 가장 많은 수 또는 가장 적은 수(각각)입니다.
Subscribers	Stage	스테이지를 구독하는 참가자의 수입니다. 활발하게 게시하고 구독하는 참여자는 게시자 및 구독자로 모두 인정됩니다. 단위: 개 유효 통계: 평균, 최대, 최소

IVS 브로드캐스트 SDK(실시간 스트리밍)

Amazon Interactive Video Service(IVS) 실시간 스트리밍 브로드캐스트 SDK는 Amazon IVS로 애플리케이션을 구축하는 개발자를 위한 것입니다. 이 SDK는 Amazon IVS 아키텍처를 활용하도록 설계되었으며, 향후 Amazon IVS와 SDK를 지속적으로 개선해나가며 새로운 기능을 추가할 예정입니다. 이 기본 브로드캐스트 SDK는 사용자가 애플리케이션에 액세스하는 데 사용하는 디바이스 및 애플리케이션에 미치는 성능 영향을 최소화하도록 설계되었습니다.

브로드캐스트 SDK는 비디오를 전송하고 수신하는 데 모두 사용된다는 점에 유의하시기 바랍니다. 즉, 호스트와 시청자에게 동일한 SDK를 사용하며 별도의 플레이어 SDK가 필요하지 않습니다.

애플리케이션에서는 다음과 같은 Amazon IVS 브로드캐스트 SDK의 주요 기능을 활용할 수 있습니다.

- 고품질 스트리밍 - 브로드캐스트 SDK는 고품질 스트리밍을 지원합니다. 카메라에서 비디오를 캡처하고 최대 720p로 인코딩합니다.
- 자동 비트 전송률 조정 - 스마트폰 사용자는 모바일을 사용하므로 브로드캐스트 전체 과정에서 네트워크 상태가 변경될 수 있습니다. Amazon IVS 브로드캐스트 SDK는 변경되는 네트워크 상태에 맞게 비디오 비트 전송률을 자동으로 조정합니다.
- 세로 및 가로 모드 지원 - 사용자가 디바이스를 어떻게 들고 있든 이미지가 똑바로 표시되고 크기가 적절하게 조정됩니다. 브로드캐스트 SDK는 세로 및 가로 캔버스 크기를 모두 지원합니다. 사용자가 디바이스를 구성된 방향에서 벗어나 회전시키는 경우 가로 세로 비율을 자동으로 관리합니다.
- 보안 스트리밍 - TLS를 사용하여 사용자의 브로드캐스트를 암호화하므로 스트림을 안전하게 보호할 수 있습니다.
- 외부 오디오 디바이스 - Amazon IVS 브로드캐스트 SDK는 오디오 잭, USB 및 Bluetooth SCO 외부 마이크를 지원합니다.

플랫폼 요구 사항:

기본 플랫폼

플랫폼	지원되는 버전
Android	9.0 이상 – 고객은 버전 5.0으로 빌드할 수 있지만 실시간 스트리밍 기능을 사용할 수 없습니다.
iOS	14 이상

IVS는 최소 4개의 주요 iOS 버전과 6개의 주요 Android 버전을 지원합니다. 현재 버전 지원은 이러한 최소 한도 이상으로 확장될 수 있습니다. 메이저 버전이 더 이상 지원되지 않을 경우 최소 3개월 전에 SDK 릴리스 노트를 통해 고객에게 알립니다.

데스크톱 브라우저

브라우저	지원되는 플랫폼	지원되는 버전
Chrome	Windows, macOS	두 가지 주요 버전(현재 및 최신 이전 버전)
Firefox	Windows, macOS	두 가지 주요 버전(현재 및 최신 이전 버전)
Edge	Windows 8.1 이상	두 가지 주요 버전(현재 및 최신 이전 버전) 엣지 레거시 제외
Safari	macOS	두 가지 주요 버전(현재 및 최신 이전 버전)

모바일 브라우저(iOS 및 Android)

브라우저	지원되는 플랫폼	지원되는 버전
Chrome	iOS, Android	두 가지 주요 버전(현재 및 최신 이전 버전)
Firefox	Android	두 가지 주요 버전(현재 및 최신 이전 버전)
Safari	iOS	두 가지 주요 버전(현재 및 최신 이전 버전)

알려진 제한 사항

- 비디오 아티팩트 및 검은색 화면 문제로 모든 모바일 디바이스에서 동시에 4명 이상의 참가자와 함께 게시 및 구독하는 것은 권장하지 않습니다. 참가자가 더 필요한 경우 오디오 전용 게시 및 구독을 구성하세요.

- 성능을 고려하고 충돌이 발생할 수 있으므로 Android 모바일 웹에서 스테이지를 합성하고 채널로 브로드캐스트하는 것은 권장하지 않습니다. 브로드캐스트 기능이 필요한 경우 IVS 실시간 스트리밍 Android 브로드캐스트 SDK를 통합하세요.

웹뷰

웹 브로드캐스트 SDK는 웹뷰 또는 웹과 유사한 환경(TV, 콘솔 등)에 대한 지원을 제공하지 않습니다. 모바일 구현에 대한 내용은 실시간 스트리밍 브로드캐스트 SDK 가이드(Android 및 iOS)를 참조하세요.

필요한 디바이스 액세스

브로드캐스트 SDK는 디바이스에 내장되어 있거나 Bluetooth, USB 또는 오디오 잭을 통해 연결되는 디바이스의 카메라와 마이크에 모두 액세스할 수 있어야 합니다.

지원

AWS는 브로드캐스트 SDK를 지속적으로 개선해나가고 있습니다. Amazon IVS 릴리즈 노트를 통해 사용 가능한 버전 및 해결된 문제를 확인해보세요. AWS Support에 문의하기 전, 브로드캐스트 SDK 버전을 적절하게 업데이트한 후 문제가 해결되었는지 확인하시기 바랍니다.

버저닝

Amazon IVS 브로드캐스트 SDK는 유의적 버저닝을 사용합니다.

이를 설명하기 위해 다음을 가정합니다.

- 최신 릴리스는 버전 4.1.3입니다.
- 이전 주요 버전의 최신 릴리스는 3.2.4입니다.
- 버전 1.x의 최신 릴리스는 1.5.6입니다.

이전 버전과 호환되는 새 기능은 최신 버전의 마이너 릴리스로 추가됩니다. 이 경우 새 기능의 다음 집합이 버전 4.2.0으로 추가됩니다.

이전 버전과 호환되는 마이너 버그 수정은 최신 버전의 패치 릴리스로 추가됩니다. 여기서 마이너 버그의 다음 수정 집합은 버전 4.1.4로 추가됩니다.

이전 버전과 호환되는 메이저 버그 수정은 다르게 처리됩니다. 이러한 버그 수정은 다음과 같이 여러 버전에 추가됩니다.

- 최신 버전의 패치 릴리스에 추가되는 경우. 이 경우 버전 4.1.4입니다.
- 이전 마이너 버전의 패치 릴리스에 추가되는 경우. 이 경우 3.2.5입니다.
- 최신 버전 1.x 릴리스의 패치 릴리스에 추가되는 경우. 이 경우 버전 1.5.7입니다.

메이저 버그 수정은 Amazon IVS 제품 팀에서 정의합니다. 일반적인 예로는 중요한 보안 업데이트와 고객에게 필요한 기타 수정이 있습니다.

참고: 위의 예시에서 확인할 수 있듯이 릴리스된 버전은 숫자를 건너뛰지 않고 순차적으로 증가합니다(예: 4.1.3에서 4.1.4로 증가). 실제로는 내부에 존재하지만 릴리스되지 않는 패치 번호가 하나 이상 있을 수 있으므로 릴리스된 버전이 4.1.3에서 4.1.6으로 증가할 수 있습니다.

IVS 브로드캐스트 SDK: 웹 가이드(실시간 스트리밍)

IVS 실시간 스트리밍 웹 브로드캐스트 SDK는 개발자에게 웹에서 대화형 실시간 환경을 구축할 수 있는 도구를 제공합니다. 이 SDK는 Amazon IVS로 웹 애플리케이션을 구축하는 개발자를 위한 것입니다.

웹 브로드캐스트 SDK를 사용하면 참가자가 비디오를 전송하고 수신할 수 있습니다. SDK에서는 다음 작업을 지원합니다.

- 스테이지 참가
- 스테이지의 다른 참가자에게 미디어 게시
- 스테이지에 있는 다른 참가자의 미디어 구독
- 스테이지에 게시된 비디오 및 오디오 관리 및 모니터링
- 각 피어 연결에 대한 WebRTC 통계 가져오기
- IVS low-latency 스트리밍 웹 브로드캐스트 SDK의 모든 작업

웹 브로드캐스트 SDK의 최신 버전: 1.8.0 (릴리스 노트)

참조 문서: Amazon IVS 웹 브로드캐스트 SDK에서 사용할 수 있는 가장 중요한 방법에 대한 자세한 내용은 https://aws.github.io/ amazon-ivs-web-broadcast /docs/sdk-reference를 참조하십시오. 가장 최신 버전의 SDK를 선택했는지 확인하세요.

샘플 코드: SDK를 빠르게 시작하려면 아래 샘플을 활용하면 좋습니다.

- HTML 및 JavaScript
- React

플랫폼 요구 사항: 지원되는 플랫폼 목록은 Amazon IVS 브로드캐스트 SDK를 참조하세요.

시작하기

가져오기

실시간의 구성 요소는 루트 브로드캐스팅 모듈이 아닌 다른 네임스페이스에 있습니다.

스크립트 태그 사용

동일한 스크립트 가져오기를 사용하면 아래 예제에 정의된 클래스와 열거형을 글로벌 객체 `IVSBroadcastClient`에서 찾을 수 있습니다.

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

npm 사용

패키지 모듈에서 클래스, 열거형 및 유형을 가져올 수도 있습니다.

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

권한 요청

앱에서 사용자의 카메라 및 마이크에 액세스할 수 있는 권한을 요청해야 하며 HTTPS를 사용하여 제공해야 합니다(이는 Amazon IVS에만 국한되지 않으며 카메라와 마이크에 액세스해야 하는 모든 웹사이트에 필요합니다).

다음은 오디오 및 비디오 장치 모두에 대한 권한을 요청하고 캡처하는 방법을 보여 주는 예제 함수입니다.

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

자세한 내용은 권한 API 및 MediaDevices.getUserMedia().를 참조하세요.

사용 가능한 장치 목록

캡처할 수 있는 기기를 확인하려면 브라우저의 MediaDevices.enumerateDevices () 메서드를 쿼리하세요.

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

디바이스에서 MediaStream 데이터 가져오기

사용 가능한 장치 목록을 가져온 후 원하는 수의 장치에서 스트림을 검색할 수 있습니다. 예를 들어 `getUserMedia()` 메서드를 사용하여 카메라에서 스트림을 검색할 수 있습니다.

스트림을 캡처할 장치를 지정하려면 미디어 제약 조건의 `audio` 또는 `video` 섹션에서 `deviceId`를 명시적으로 설정할 수 있습니다. 또는 `deviceId`를 생략하고 사용자가 브라우저 프롬프트에서 장치를 선택하도록 할 수 있습니다.

`width` 및 `height` 제약 조건을 사용하여 이상적인 카메라 해상도를 지정할 수도 있습니다(이러한 제약 조건에 대한 자세한 내용은 여기를 참조하세요). SDK는 최대 방송 해상도에 해당하는 너비 및 높이 제한을 자동으로 적용합니다. 하지만 SDK에 소스를 추가한 후에 소스 종횡비가 변경되지 않도록 이를 직접 적용하는 것도 좋습니다.

실시간 스트리밍의 경우 미디어가 720p 해상도로 제한되는지 확인하십시오. 특히 폭과 높이에 대한 getDisplayMedia 제한 getUserMedia 및 제한 값을 곱한 경우 921600 (1280*720) 을 초과해서는 안 됩니다.

```
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 },
});
```

게시 및 구독

개념

실시간 기능의 3가지 핵심 개념은 스테이지, 전략 및 이벤트입니다. 설계 목표는 작동하는 제품을 구축하는 데 필요한 클라이언트 측 로직의 수를 최소화하는 것입니다.

단계

Stage 클래스는 호스트 애플리케이션과 SDK 사이의 주요 상호 작용 지점입니다. 스테이지 자체를 나타내며 스테이지에 참가하고 나가는 데 사용됩니다. 스테이지를 만들고 참가하려면 제어 플레인에서 유효하고 만료되지 않은 토큰 문자열(token으로 표시됨)이 필요합니다. 스테이지 참가 및 나가기는 간단합니다.

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

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

stage.leave();
```

Strategy

StageStrategy 인터페이스는 호스트 애플리케이션이 원하는 스테이지 상태를 SDK에 전달하는 방법을 제공합니다. shouldSubscribeToParticipant, shouldPublishParticipant 및 stageStreamsToPublish 함수를 구현해야 합니다. 모든 함수를 아래에서 설명합니다.

정의된 전략을 사용하려면 Stage 생성자에게 전달합니다. 다음은 참가자의 웹캠을 스테이지에 게시하고 모든 참가자를 구독하는 전략을 사용하는 애플리케이션의 전체 예제입니다. 각 필수 전략 함수의 목적은 후속 섹션에서 자세히 설명합니다.

```
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();
```

참가자 구독

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

원격 참가자가 스테이지에 참가하면 SDK는 호스트 애플리케이션에 해당 참가자의 원하는 구독 상태를 쿼리합니다. 옵션은 NONE, AUDIO_ONLY 및 AUDIO_VIDEO입니다. 이 함수의 값을 반환할 때 호스트 애플리케이션은 게시 상태, 현재 구독 상태 또는 스테이지 연결 상태에 대해 걱정할 필요가 없습니다. AUDIO_VIDEO가 반환되는 경우 SDK는 구독 전에 원격 참가자가 게시할 때까지 기다리고, 프로세스 전반에 걸쳐 이벤트를 발생시켜 호스트 애플리케이션을 업데이트합니다.

다음은 샘플 구현입니다.

```
const strategy = {

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

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

항상 모든 참가자가 서로 보기를 원하는 호스트 애플리케이션(예: 비디오 채팅 애플리케이션)을 위한 이 기능의 전체 구현입니다.

고급 구현도 가능합니다. `ParticipantInfo`의 `userInfo` 속성을 사용하여 서버에서 제공하는 특성을 기반으로 참가자를 선택적으로 구독합니다.

```
const strategy = {

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

이를 통해 중재자가 직접 보거나 듣지 않고 모든 게스트를 모니터링할 수 있는 스테이지를 만들 수 있습니다. 호스트 애플리케이션은 추가 비즈니스 로직을 사용하여 중재자가 서로를 볼 수는 있지만 게스트에게는 보이지 않도록 할 수 있습니다.

게시

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

스테이지에 연결되면 SDK는 호스트 애플리케이션을 쿼리하여 특정 참가자가 게시해야 하는지 여부를 확인합니다. 이는 제공된 토큰을 기반으로 게시할 권한이 있는 로컬 참가자에게만 호출됩니다.

다음은 샘플 구현입니다.

```
const strategy = {

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

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

이는 사용자가 항상 게시하기를 원하는 표준 비디오 채팅 애플리케이션에 대한 구현입니다. 오디오 및 비디오를 음소거 또는 음소거 해제하여 즉시 숨기거나 보기/듣기가 가능하도록 할 수 있습니다. (게시/게시 취소를 사용할 수도 있지만 속도가 훨씬 느립니다. 음소거/음소거 해제는 가시성을 자주 변경하는 것이 바람직한 사용 사례에 적합합니다.)

게시할 스트림 선택

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

게시할 때 이를 사용하여 게시해야 하는 오디오 및 비디오 스트림을 결정합니다. 이에 대해서는 미디어 스트림 게시에서 자세히 설명합니다.

전략 업데이트

전략은 동적이어야 합니다. 위 함수 중에서 반환되는 값은 언제든지 변경될 수 있습니다. 예를 들어 호스트 애플리케이션이 최종 사용자가 버튼을 탭할 때까지 게시하지 않으려는 경우, shouldPublishParticipant에서 변수를 반환할 수 있습니다(예: hasUserTappedPublishButton). 최종 사용자의 상호 작용에 따라 변수가 변경되면 stage.refreshStrategy()를 호출하여 SDK에 최신 값에 대한 전략을 쿼리하고 변경된 사항만 적용하도록 신호를 보냅니다. SDK에서 shouldPublishParticipant 값이 변경된 것을 관찰하면 게시 프로세스가 시작됩니다. SDK 쿼리와 모든 함수가 이전과 동일한 값을 반환하는 경우 refreshStrategy 호출 시 스테이지가 수정되지 않습니다.

shouldSubscribeToParticipant의 반환 값이 AUDIO_VIDEO에서 AUDIO_ONLY로 변경된 경우 이전에 비디오 스트림이 존재했다면 반환된 값이 변경된 모든 참가자에 대한 비디오 스트림이 제거됩니다.

일반적으로 스테이지는 전략을 사용하여 이전 전략과 현재 전략 간의 차이를 가장 효율적으로 적용하므로 호스트 애플리케이션에서 이를 올바르게 관리하는 데 필요한 모든 상태에 대해 걱정할 필요가 없습니다. 이로 인해 stage.refreshStrategy() 호출은 전략이 변경되지 않는 한 아무 소용도 없으므로 소모량이 적은 작업이라고 생각하면 됩니다.

이벤트

Stage 인스턴스는 이벤트 이미터입니다. stage.on()을 사용하면 스테이지 상태가 프로토콜은 호스트 애플리케이션에 전달됩니다. 호스트 애플리케이션의 UI 업데이트는 전적으로 이벤트에 의해 지원될 수 있습니다. 이벤트는 다음과 같습니다.

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

대부분의 이벤트에서 해당 `ParticipantInfo`가 제공됩니다.

이벤트에서 제공하는 정보가 전략의 반환 값에 영향을 미칠 것으로 예상되지는 않습니다. 예를 들어, `shouldSubscribeToParticipant`의 반환 값은 `STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED`가 호출될 때 변경되지 않을 것으로 예상됩니다. 호스트 애플리케이션이 특정 참가자를 구독하려는 경우 해당 참가자의 게시 상태와 무관하게 원하는 구독 유형을 반환해야 합니다. SDK는 원하는 전략 상태가 스테이지 상태를 기반을 정확한 시간에 실행되도록 하는 역할을 합니다.

미디어 스트림 게시

마이크 및 카메라와 같은 로컬 장치는 위의 장치에서 검색에서 설명한 것과 동일한 단계를 사용하여 검색합니다. MediaStream 이 예제에서는 `MediaStream`을 사용하여 SDK에서 게시하는 데 사용되는 `LocalStageStream` 객체 목록을 만듭니다.

```
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
    }
}
```

화면 공유 게시

애플리케이션에서 종종 사용자의 웹 카메라 외에도 화면 공유를 게시해야 합니다. 화면 공유를 게시하려면 고유한 토큰으로 추가 `Stage`를 만들어야 합니다.

```
// 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();
```

참가자 표시 및 제거

구독이 완료되면 STAGE_PARTICIPANT_STREAMS_ADDED 이벤트를 통해 StageStream 객체 배열을 받게 됩니다. 이벤트는 미디어 스트림을 표시할 때 도움이 되는 참가자 정보도 제공합니다.

```
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));
})
```

참가자가 게시를 중단하거나 스트림 구독이 취소되면 제거된 스트림과 함께 STAGE_PARTICIPANT_STREAMS_REMOVED 함수가 호출됩니다. 호스트 애플리케이션은 이를 신호로 사용하여 DOM에서 참가자의 비디오 스트림을 제거해야 합니다.

STAGE_PARTICIPANT_STREAMS_REMOVED는 다음을 포함하여 스트림이 제거될 수 있는 모든 시나리오에서 호출됩니다.

- 원격 참가자가 게시를 중단합니다.
- 로컬 디바이스가 구독을 취소하거나 구독을 AUDIO_VIDEO에서 AUDIO_ONLY로 변경합니다.
- 원격 참가자가 스테이지를 나갑니다.
- 로컬 참가자가 스테이지를 나갑니다.

모든 시나리오에서 STAGE_PARTICIPANT_STREAMS_REMOVED가 호출되므로 원격 또는 로컬 나가기 작업 중에 UI에서 참가자를 제거하는 사용자 지정 비즈니스 로직이 필요하지 않습니다.

미디어 스트림 음소거 및 음소거 해제

LocalStageStream 객체에는 스트림의 음소거 여부를 제어하는 setMuted 함수가 있습니다. 이 함수는 stageStreamsToPublish 전략 함수에서 반환되기 전이나 후에 스트림에서 호출할 수 있습니다.

중요: refreshStrategy 호출 이후 stageStreamsToPublish에 의해 새 LocalStageStream 객체 인스턴스가 반환되면 새 스트림 객체의 음소거 상태가 스테이지에 적용됩니다. 새 LocalStageStream 인스턴스를 만들 때는 예상되는 음소거 상태가 유지되도록 주의해야 합니다.

원격 참가자 미디어 음소거 상태 모니터링

참가자가 비디오 또는 오디오의 음소거 상태를 변경하면 변경된 스트림 목록과 함께 STAGE_STREAM_MUTE_CHANGED 이벤트가 트리거됩니다. StageStream의 isMuted 속성을 사용하여 UI를 적절히 업데이트합니다.

```
stage.on(StageEvents.STAGE_STREAM_MUTE_CHANGED, (participant, stream) => {
   if (stream.streamType === 'video' && stream.isMuted) {
```

```
        // handle UI changes for video track getting muted
    }
})
```

또한 오디오 또는 비디오가 음소거되었는지 여부에 StageParticipantInfo대한 상태 정보를 확인할 수 있습니다.

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

WebRTC 통계 가져오기

게시 스트림 또는 구독 스트림에 대한 최신 WebRTC 통계를 가져오려면 StageStream에서 getStats를 사용합니다. await을 통해 또는 promise 체인으로 통계를 검색할 수 있는 비동기 메서드입니다. 결과는 모든 표준 통계를 포함하는 딕셔너리인 RTCStatsReport입니다.

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

미디어 최적화

최상의 성능을 위해 다음 제약 조건으로 getUserMedia 및 getDisplayMedia 호출을 제한하는 것이 좋습니다.

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

LocalStageStream 생성자에 전달된 추가 옵션을 통해 미디어를 더 제한할 수 있습니다.

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

위의 코드에서

- `minBitrate`는 브라우저가 사용할 것으로 예상되는 최소 비트레이트를 설정합니다. 그러나 조금 복잡한 비디오 스트림은 인코더를 이 비트레이트보다 낮게 만들 수 있습니다.
- `maxBitrate`는 브라우저가 이 스트림에 대해 초과하지 않을 것으로 예상되는 최대 비트레이트를 설정합니다.
- `maxFramerate`는 브라우저가 이 스트림에 대해 초과하지 않을 것으로 예상되는 최대 프레임 속도를 설정합니다.
- `simulcast` 옵션은 Chromium 기반 브라우저에서만 사용할 수 있습니다. 이 옵션을 사용하면 스트림의 3개 렌더링 계층을 전송할 수 있습니다.
  - 이를 통해 서버는 네트워킹 제한에 따라 다른 참가자에게 전송할 변환을 선택할 수 있습니다.
  - `simulcast`가 `maxBitrate` 및/또는 `maxFramerate` 값과 함께 지정되는 경우 `maxBitrate`가 내부 SDK의 두 번째로 높은 계층의 기본 `maxBitrate` 값인 900kbps 아래로 내려가지 않는 한 이러한 값을 염두에 두고 가장 높은 변환 계층이 구성될 것으로 예상됩니다.
  - `maxBitrate`가 두 번째로 높은 계층의 기본값에 비해 너무 낮게 지정되면 `simulcast`가 비활성화됩니다.
  - `shouldPublishParticipant`가 `false`를 반환하고, `refreshStrategy`를 직접적으로 호출하고, `shouldPublishParticipant`가 `true`를 반환하게 하고, `refreshStrategy`를 다시 직접적으로 호출하는 조합을 통해 미디어를 다시 게시하지 않고는 `simulcast`를 켜고 끌 수 없습니다.

참가자 특성 가져오기

`CreateParticipantToken` 엔드포인트 요청에서 특성을 지정하는 경우 `StageParticipantInfo` 속성에서 특성을 볼 수 있습니다.

```
stage.on(StageEvents.STAGE_PARTICIPANT_JOINED, (participant) => {
```

```
    console.log(`Participant ${participant.id} info:`, participant.attributes);
})
```

네트워크 문제 처리

로컬 디바이스의 네트워크 연결이 끊어지면 SDK는 사용자 작업 없이 내부적으로 재연결을 시도합니다. SDK에서 재연결이 성공적이지 않아 사용자 작업이 필요한 경우도 있습니다.

스테이지의 상태는 대체로 STAGE_CONNECTION_STATE_CHANGED 이벤트를 통해 처리할 수 있습니다.

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

일반적으로 스테이지에 성공적으로 참가한 후 오류가 발생하면 SDK 연결이 끊어진 후 연결 재설정에 실패했음을 나타냅니다. 새 Stage 객체를 만들고 네트워크 상태가 개선되면 참가를 시도합니다.

IVS 채널로 스테이지 브로드캐스트

스테이지를 브로드캐스트하려면 별도의 IVSBroadcastClient 세션을 만든 다음 위에서 설명한 대로 SDK로 브로드캐스트하기 위한 일반적인 지침을 따릅니다. STAGE_PARTICIPANT_STREAMS_ADDED를 통해 노출된 StageStream 목록을 사용하여 다음과 같이 브로드캐스트 스트림 구성에 적용할 수 있는 참가자 미디어 스트림을 검색할 수 있습니다.

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

필요에 따라 스테이지를 합성하고 IVS 지연 시간이 짧은 채널로 브로드캐스트하여 더 많은 청중에게 다가갈 수 있습니다. IVS 지연 시간이 짧은 스트리밍 사용 설명서의 Amazon IVS 스트림에서 여러 호스트 활성화를 참조하세요.

알려진 문제 및 해결 방법

- `stage.leave()`를 호출하지 않고 브라우저 탭을 닫거나 브라우저를 종료해도 사용자 세션에 최대 10초 동안 정지된 프레임이나 검은색 화면이 표시될 수 있습니다.

  해결 방법: 없음

- Safari 세션은 세션 시작 후 참가하는 사용자에게 간헐적으로 검은색 화면으로 표시됩니다.

  해결 방법: 브라우저를 새로 고치고 세션을 다시 연결합니다.

- Safari가 네트워크 전환에서 정상적으로 복구되지 않습니다.

  해결 방법: 브라우저를 새로 고치고 세션을 다시 연결합니다.

- 개발자 콘솔에서 `Error: UnintentionalError at StageSocket.onClose` 오류가 반복됩니다.

  해결 방법: 참가자 토큰당 하나의 스테이지만 생성할 수 있습니다. 이 오류는 인스턴스가 한 디바이스에 있거나 여러 디바이스에 있는지 여부와 무관하게 동일한 참가자 토큰으로 둘 이상의 `Stage` 인스턴스가 생성될 때 발생합니다.

- 상태를 유지하는 데 문제가 있을 수 있으며 `StageEvents.STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED` 이벤트를 들을 때 `StageParticipantPublishState.PUBLISHED` `StageParticipantPublishState.ATTEMPTING_PUBLISH` 상태가 반복될 수 있습니다.

  해결 방법: 또는 를 호출할 때 비디오 해상도를 720p로 제한하십시오. `getUserMedia` `getDisplayMedia` 특히 너비와 높이의 `getDisplayMedia` 제한 `getUserMedia` 및 제한 값을 곱한 경우 921600 (1280*720) 을 초과해서는 안 됩니다.

Safari 제한 사항

- 권한 프롬프트를 거부하려면 OS 수준에서 Safari 웹 사이트 설정의 권한을 재설정해야 합니다.
- Safari는 기본적으로 모든 장치를 Firefox나 Chrome만큼 효과적으로 감지하지는 않습니다. 예를 들어 OBS 가상 카메라는 감지되지 않습니다.

Firefox 제한 사항

- Firefox에서 화면을 공유하려면 시스템 권한을 활성화해야 합니다. 활성화한 후에는 사용자가 Firefox를 다시 시작해야 제대로 작동합니다. 그렇지 않으면 권한이 차단된 것으로 인식되면 브라우저에서 예외가 발생합니다. NotFoundError
- `getCapabilities` 메서드가 없습니다. 즉, 사용자는 미디어 트랙의 해상도나 종횡비를 얻을 수 없습니다. 이 Bugzilla 타래를 참조하세요.
- 몇 가지 `AudioContext` 속성이 없습니다(예: 지연 시간 및 채널 수). 이는 오디오 트랙을 조작하려는 고급 사용자에게는 문제가 될 수 있습니다.

- MacOS에서는 `getUserMedia`의 카메라 피드가 4:3 종횡비로 제한됩니다. Bugzilla 스레드 1과 Bugzilla 스레드 2를 참조하세요.

- `getDisplayMedia`에서는 오디오 캡처가 지원되지 않습니다. 이 Bugzilla 타래를 참조하세요.
- 화면 캡처의 프레임 속도가 최적이 아닙니다(약 15fps?). 이 Bugzilla 타래를 참조하세요.

모바일 웹 제한 사항

- getDisplayMedia모바일 장치에서는 화면 공유가 지원되지 않습니다.

해결 방법: 없음

- `leave()`을 호출하지 않고 브라우저를 닫으면 참가자가 나가기까지 15초~30초가 걸립니다.

  해결 방법: 사용자가 연결을 제대로 끊도록 유도하는 UI를 추가합니다.

- 앱을 백그라운드로 전환하면 동영상 게시가 중지됩니다.

  해결 방법: 게시자가 일시 중지된 경우 UI 슬레이트를 표시합니다.

- Android 디바이스에서 카메라 음소거를 해제한 후 약 5초 동안 동영상 프레임 속도가 떨어집니다.

  해결 방법: 없음

- iOS 16.0의 경우 동영상 피드가 회전하면서 늘어납니다.

  해결 방법: 이 알려진 OS 문제를 설명하는 UI를 표시합니다.

- 오디오 입력 디바이스를 전환하면 오디오 출력 디바이스가 자동으로 전환됩니다.

  해결 방법: 없음

- 브라우저를 백그라운드로 만들면 게시 스트림이 검게 변하고 오디오만 출력됩니다.

  해결 방법: 없음 이는 보안상의 이유 때문입니다.

오류 처리

이 섹션에서는 오류 조건, 웹 브로드캐스트 SDK가 애플리케이션에 오류를 보고하는 방법, 이러한 오류가 발생할 경우 애플리케이션이 수행해야 하는 작업에 대한 개요를 다룹니다. 오류에는 다음과 같은 네 가지 범주가 있습니다.

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

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

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

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

스테이지 인스턴스화 오류

스테이지 인스턴스화는 토큰을 원격으로 검증하지는 않지만 클라이언트 측에서 검증할 수 있는 몇 가지 기본적인 토큰 문제를 확인합니다. 따라서 SDK에서 오류가 발생할 수 있습니다.

잘못된 참가자 토큰

스테이지 토큰의 형식이 잘못된 경우 발생합니다. 스테이지를 인스턴스화할 때 SDK에서 “스테이지 토큰을 파싱하는 중 오류가 발생했습니다.”라는 메시지와 함께 오류가 발생합니다.

조치: 유효한 토큰을 생성한 후 인스턴스화를 다시 시도해 보세요.

스테이지 참가 오류

처음 스테이지에 참가하려고 할 때 발생할 수 있는 오류입니다.

스테이지 삭제됨

삭제된 스테이지(토큰과 연결됨)에 참가할 때 발생합니다. `join`SDK 메서드에서 “10초 InitialConnectTimedOut 후”라는 메시지와 함께 오류가 발생합니다.

조치: 새 스테이지로 유효한 토큰을 생성한 후 다시 참가해 보세요.

만료된 참가자 토큰

토큰이 만료될 때 발생합니다. `join` SDK 메서드에서 “토큰이 만료되어 더 이상 유효하지 않습니다.”라는 메시지와 함께 오류가 발생합니다.

조치: 새 토큰을 생성한 후 다시 참가해 보세요.

유효하지 않거나 취소된 참가자 토큰

토큰이 유효하지 않거나 취소/연결이 끊겼을 때 발생합니다. join SDK 메서드에서 “10초 InitialConnectTimedOut 후”라는 메시지와 함께 오류가 발생합니다.

조치: 새 토큰을 생성한 후 다시 참가해 보세요.

연결이 끊긴 토큰

스테이지 토큰의 형식이 잘못되진 않았지만 스테이지 서버에서 거부된 경우에 발생합니다. join SDK 메서드에서 “10초 InitialConnectTimedOut 후”라는 메시지와 함께 오류가 발생합니다.

조치: 유효한 토큰을 생성한 후 다시 참가해 보세요.

첫 참가 시 네트워크 오류

SDK가 스테이지 서버에 접속하여 연결을 설정할 수 없는 경우 발생합니다. join SDK 메서드에서 “10초 InitialConnectTimedOut 후”라는 메시지와 함께 오류가 발생합니다.

조치: 디바이스 연결이 복구될 때까지 기다린 후 다시 참가해 보세요.

이미 참가한 경우 네트워크 오류

디바이스의 네트워크 연결이 끊어지면 SDK와 스테이지 서버 연결이 끊어질 수 있습니다. SDK가 더 이상 백엔드 서비스에 연결할 수 없기 때문에 콘솔에 오류가 표시될 수 있습니다. https://broadcast.stats.live-video.net에 대한 POST는 실패합니다.

게시 및/또는 구독 중인 경우 콘솔에 게시/구독 시도와 관련된 오류가 표시됩니다.

내부적으로 SDK는 지수 백오프 전략을 사용하여 재연결을 시도합니다.

조치: 디바이스 연결이 복구될 때까지 기다리세요. 게시 또는 구독 중인 경우 전략을 새로 고쳐 미디어 스트림을 다시 게시하세요.

게시 및 구독 오류

게시 오류: 게시 상태

게시가 실패하면 SDK가 ERRORED를 보고합니다. 이는 네트워크 상태 또는 게시자 수용량이 가득 찬 스테이지 때문에 발생할 수 있습니다.

```
stage.on(StageEvents.STAGE_PARTICIPANT_PUBLISH_STATE_CHANGED, (participantInfo, state)
 => {
```

```
  if (state === StageParticipantPublishState.ERRORED) {
      // Handle
  }
});
```

조치: 전략을 새로 고쳐 미디어 스트림을 다시 게시하세요.

구독 오류

구독이 실패하면 SDK가 ERRORED를 보고합니다. 이는 네트워크 상태 또는 구독자 수용량이 가득 찬 스테이지 때문에 발생할 수 있습니다.

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

조치: 전략을 새로 고쳐 새로운 구독을 시도하세요.

IVS 브로드캐스트 SDK: Android 가이드(실시간 스트리밍)

IVS 실시간 스트리밍 Android 브로드캐스트 SDK를 사용하면 참가자가 Android에서 비디오를 전송하고 수신할 수 있습니다.

`com.amazonaws.ivs.broadcast` 패키지는 본 문서에서 설명하는 인터페이스를 구현합니다. SDK에서는 다음 작업을 지원합니다.

- 스테이지 참가
- 스테이지의 다른 참가자에게 미디어 게시
- 스테이지에 있는 다른 참가자의 미디어 구독
- 스테이지에 게시된 비디오 및 오디오 관리 및 모니터링
- 각 피어 연결에 대한 WebRTC 통계 가져오기
- IVS 지연 시간이 짧은 스트리밍 Android 브로드캐스트 SDK의 모든 작업

안드로이드 브로드캐스트 SDK 최신 버전: 1.14.1 (릴리스 노트)

참조 문서: Amazon IVS Android 브로드캐스트 SDK에서 사용할 수 있는 가장 중요한 방법에 대한 자세한 내용은 https://aws.github.io/ amazon-ivs-broadcast-docs /1.14.1/android/의 참조 설명서를 참조하십시오.

샘플 코드: https://github.com/aws-samples/ -sample의 Android 샘플 리포지토리를 참조하십시오. GitHub amazon-ivs-broadcast-android

플랫폼 요구 사항: Android 9.0 이상.

시작하기

라이브러리 설치

Amazon 개발 환경에 Amazon IVS Android 브로드캐스트 라이브러리를 추가하려면 여기에 나온 것처럼(최신 버전의 Amazon IVS 브로드캐스트 SDK용) 라이브러리를 모듈의 `build.gradle` 파일에 추가합니다.

```
repositories {
    mavenCentral()
}

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

매니페스트에 다음 권한을 추가하여 SDK에서 스피커폰을 활성화 및 비활성화할 수 있도록 허용합니다.

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

또는 SDK를 수동으로 설치하려면 다음 위치에서 최신 버전을 다운로드하세요.

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

`-stages`가 추가된 `aar`를 다운로드해야 합니다.

권한 요청

앱에서 사용자의 카메라 및 마이크에 액세스할 수 있는 권한을 요청해야 합니다. (이는 Amazon IVS에만 국한되지 않으며 카메라와 마이크에 액세스해야 하는 모든 애플리케이션에 필요합니다.)

여기에서는 사용자가 부여된 권한이 이미 있는지 확인하고, 그렇지 않으면 권한을 요청합니다.

```
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;
    }
}
```

여기서는 다음과 같은 사용자 응답을 받습니다.

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

게시 및 구독

개념

실시간 기능의 3가지 핵심 개념은 스테이지, 전략 및 렌더러입니다. 설계 목표는 작동하는 제품을 구축하는 데 필요한 클라이언트 측 로직의 수를 최소화하는 것입니다.

단계

Stage 클래스는 호스트 애플리케이션과 SDK 사이의 주요 상호 작용 지점입니다. 스테이지 자체를 나타내며 스테이지에 참가하고 나가는 데 사용됩니다. 스테이지를 만들고 참가하려면 제어 플레인에서 유효하고 만료되지 않은 토큰 문자열(token으로 표시됨)이 필요합니다. 스테이지 참가 및 나가기는 간단합니다.

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

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

stage.leave();
```

Stage 클래스는 StageRenderer가 첨부될 수 있는 위치이기도 합니다.

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

Strategy

Stage.Strategy 인터페이스는 호스트 애플리케이션이 원하는 스테이지 상태를 SDK에 전달하는 방법을 제공합니다. shouldSubscribeToParticipant, shouldPublishFromParticipant 및 stageStreamsToPublishForParticipant 함수를 구현해야 합니다. 모든 함수를 아래에서 설명합니다.

참가자 구독

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

원격 참가자가 스테이지에 참가하면 SDK는 호스트 애플리케이션에 해당 참가자의 원하는 구독 상태를 쿼리합니다. 옵션은 NONE, AUDIO_ONLY 및 AUDIO_VIDEO입니다. 이 함수의 값을 반환할 때 호스트 애플리케이션은 게시 상태, 현재 구독 상태 또는 스테이지 연결 상태에 대해 걱정할 필요가 없습니다. AUDIO_VIDEO가 반환되는 경우 SDK는 구독 전에 원격 참가자가 게시할 때까지 기다리고, 프로세스 전반에 걸쳐 렌더러를 통해 호스트 애플리케이션을 업데이트합니다.

다음은 샘플 구현입니다.

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

항상 모든 참가자가 서로 보기를 원하는 호스트 애플리케이션(예: 비디오 채팅 애플리케이션)을 위한 이 기능의 전체 구현입니다.

고급 구현도 가능합니다. ParticipantInfo의 userInfo 속성을 사용하여 서버에서 제공하는 특성을 기반으로 참가자를 선택적으로 구독합니다.

```
@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;
 }
}
```

이를 통해 중재자가 직접 보거나 듣지 않고 모든 게스트를 모니터링할 수 있는 스테이지를 만들 수 있습니다. 호스트 애플리케이션은 추가 비즈니스 로직을 사용하여 중재자가 서로를 볼 수는 있지만 게스트에게는 보이지 않도록 할 수 있습니다.

게시

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

스테이지에 연결되면 SDK는 호스트 애플리케이션을 쿼리하여 특정 참가자가 게시해야 하는지 여부를 확인합니다. 이는 제공된 토큰을 기반으로 게시할 권한이 있는 로컬 참가자에게만 호출됩니다.

다음은 샘플 구현입니다.

```
@Override
```

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

이는 사용자가 항상 게시하기를 원하는 표준 비디오 채팅 애플리케이션에 대한 구현입니다. 오디오 및 비디오를 음소거 또는 음소거 해제하여 즉시 숨기거나 보기/듣기가 가능하도록 할 수 있습니다. (게시/게시 취소를 사용할 수도 있지만 속도가 훨씬 느립니다. 음소거/음소거 해제는 가시성을 자주 변경하는 것이 바람직한 사용 사례에 적합합니다.)

게시할 스트림 선택

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

게시할 때 이를 사용하여 게시해야 하는 오디오 및 비디오 스트림을 결정합니다. 이에 대해서는 미디어 스트림 게시에서 자세히 설명합니다.

전략 업데이트

전략은 동적이어야 합니다. 위 함수 중에서 반환되는 값은 언제든지 변경될 수 있습니다. 예를 들어 호스트 애플리케이션이 최종 사용자가 버튼을 탭할 때까지 게시하지 않으려는 경우, `shouldPublishFromParticipant`에서 변수를 반환할 수 있습니다(예: `hasUserTappedPublishButton`). 최종 사용자의 상호 작용에 따라 변수가 변경되면 `stage.refreshStrategy()`를 호출하여 SDK에 최신 값에 대한 전략을 쿼리하고 변경된 사항만 적용하도록 신호를 보냅니다. SDK에서 `shouldPublishFromParticipant` 값이 변경된 것을 관찰하면 게시 프로세스가 시작됩니다. SDK 쿼리와 모든 함수가 이전과 동일한 값을 반환하는 경우 `refreshStrategy` 호출 시 스테이지가 수정되지 않습니다.

`shouldSubscribeToParticipant`의 반환 값이 `AUDIO_VIDEO`에서 `AUDIO_ONLY`로 변경된 경우 이전에 비디오 스트림이 존재했다면 반환된 값이 변경된 모든 참가자에 대한 비디오 스트림이 제거됩니다.

일반적으로 스테이지는 전략을 사용하여 이전 전략과 현재 전략 간의 차이를 가장 효율적으로 적용하므로 호스트 애플리케이션에서 이를 올바르게 관리하는 데 필요한 모든 상태에 대해 걱정할 필요가 없습니다. 이로 인해 `stage.refreshStrategy()` 호출은 전략이 변경되지 않는 한 아무 소용도 없으므로 소모량이 적은 작업이라고 생각하면 됩니다.

렌더러

StageRenderer 인터페이스는 호스트 애플리케이션에 스테이지 상태를 전달하는 방법을 제공합니다. 호스트 애플리케이션의 UI 업데이트는 대체로 렌더러에서 제공하는 이벤트에 의해 전적으로 이루어질 수 있습니다. 렌더러는 다음과 같은 함수를 제공합니다.

```
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);
```

대부분의 메서드에서 해당 Stage 및 ParticipantInfo가 제공됩니다.

렌더러에서 제공하는 정보가 전략의 반환 값에 영향을 미칠 것으로 예상되지는 않습니다. 예를 들어, shouldSubscribeToParticipant의 반환 값은 onParticipantPublishStateChanged가 호출될 때 변경되지 않을 것으로 예상됩니다. 호스트 애플리케이션이 특정 참가자를 구독하려는 경우 해당 참가자의 게시 상태와 무관하게 원하는 구독 유형을 반환해야 합니다. SDK는 원하는 전략 상태가 스테이지 상태를 기반을 정확한 시간에 실행되도록 하는 역할을 합니다.

StageRenderer는 스테이지 클래스에 연결될 수 있습니다.

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

게시 참가자만 onParticipantJoined를 트리거하고, 참가자가 게시를 중단하거나 스테이지 세션에서 나갈 때마다 onParticipantLeft가 트리거됩니다.

미디어 스트림 게시

내장 마이크 및 카메라와 같은 로컬 디바이스는 DeviceDiscovery를 통해 검색됩니다. 다음은 전면 카메라와 기본 마이크를 선택한 다음 SDK에 게시될 LocalStageStreams로 반환하는 예제입니다.

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

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

Device frontCamera = null;
Device microphone = null;

// Create streams using the front camera, first microphone
for (Device device : devices) {
 Device.Descriptor descriptor = device.getDescriptor();
 if (!frontCamera && descriptor.type == Device.Descriptor.DeviceType.Camera &&
 descriptor.position = Device.Descriptor.Position.FRONT) {
  front Camera = device;
 }
 if (!microphone && descriptor.type == Device.Descriptor.DeviceType.Microphone) {
  microphone = device;
 }
}

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

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

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

참가자 표시 및 제거

구독이 완료되면 렌더러의 onStreamsAdded 함수를 통해 StageStream 객체 배열을 받게 됩니다. ImageStageStream에서 미리 보기를 검색할 수 있습니다.

```
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);
```

AudioStageStream에서 오디오 수준 통계를 검색할 수 있습니다.

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

참가자가 게시를 중단하거나 참가자 구독이 취소되면 제거된 스트림과 함께 onStreamsRemoved 함수가 호출됩니다. 호스트 애플리케이션은 이를 신호로 사용하여 보기 계층 구조에서 참가자의 비디오 스트림을 제거해야 합니다.

onStreamsRemoved는 다음을 포함하여 스트림이 제거될 수 있는 모든 시나리오에서 호출됩니다.

- 원격 참가자가 게시를 중단합니다.
- 로컬 디바이스가 구독을 취소하거나 구독을 AUDIO_VIDEO에서 AUDIO_ONLY로 변경합니다.
- 원격 참가자가 스테이지를 나갑니다.
- 로컬 참가자가 스테이지를 나갑니다.

모든 시나리오에서 onStreamsRemoved가 호출되므로 원격 또는 로컬 나가기 작업 중에 UI에서 참가자를 제거하는 사용자 지정 비즈니스 로직이 필요하지 않습니다.

미디어 스트림 음소거 및 음소거 해제

LocalStageStream 객체에는 스트림의 음소거 여부를 제어하는 setMuted 함수가 있습니다. 이 함수는 streamsToPublishForParticipant 전략 함수에서 반환되기 전이나 후에 스트림에서 호출할 수 있습니다.

중요: refreshStrategy 호출 이후 streamsToPublishForParticipant에 의해 새 LocalStageStream 객체 인스턴스가 반환되면 새 스트림 객체의 음소거 상태가 스테이지에 적용됩니다. 새 LocalStageStream 인스턴스를 만들 때는 예상되는 음소거 상태가 유지되도록 주의해야 합니다.

원격 참가자 미디어 음소거 상태 모니터링

참가자가 비디오 또는 오디오 스트림의 음소거 상태를 변경할 때 변경된 스트림 목록과 함께 렌더러 onStreamMutedChanged 함수가 호출됩니다. StageStream의 getMuted 메서드를 사용하여 UI를 적절히 업데이트합니다.

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

WebRTC 통계 가져오기

게시 스트림 또는 구독 스트림에 대한 최신 WebRTC 통계를 가져오려면 StageStream에서 requestRTCStats를 사용합니다. 수집이 완료되면 StageStream에서 설정할 수 있는 StageStream.Listener를 통해 통계를 받습니다.

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

참가자 특성 가져오기

CreateParticipantToken 엔드포인트 요청에서 특성을 지정하는 경우 ParticipantInfo 속성에서 특성을 볼 수 있습니다.

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

백그라운드에서 세션 계속하기

앱이 백그라운드로 전환되면 게시를 중단하거나 다른 원격 참가자의 오디오만 구독하고 싶을 수 있습니다. 이렇게 하려면 Strategy 구현을 업데이트하여 게시를 중단하고 AUDIO_ONLY를 구독합니다(또는 해당하는 경우 NONE).

```
// Local variables before going into the background
boolean shouldPublish = true;
Stage.SubscribeType subscribeType = Stage.SubscribeType.AUDIO_VIDEO;

// Stage.Strategy implementation
@Override
boolean shouldPublishFromParticipant(@NonNull Stage stage, @NonNull ParticipantInfo
 participantInfo) {
 return shouldPublish;
}

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

// In our Activity, modify desired publish/subscribe when we go to background, then
 call refreshStrategy to update the stage
@Override
void onStop() {
 super.onStop();
 shouldPublish = false;
```

```
 subscribeTpye = Stage.SubscribeType.AUDIO_ONLY;
 stage.refreshStrategy();
}
```

동시 방송을 사용한 계층화된 인코딩 활성화/비활성화

미디어 스트림을 게시할 때 SDK는 고품질 및 저품질 비디오 스트림을 전송하므로 원격 참가자는 다운링크 대역폭이 제한되어 있어도 스트림을 구독할 수 있습니다. 동시 방송을 사용한 계층화된 인코딩이 기본적으로 켜져 있습니다. `StageVideoConfiguration.Simulcast` 클래스를 사용하여 이를 비활성화할 수 있습니다.

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

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera, config);

// Other Stage implementation code
```

비디오 구성 제한

SDK는 `StageVideoConfiguration.setSize(BroadcastConfiguration.Vec2 size)`를 사용하는 세로 모드 또는 가로 모드 강제 사용을 지원하지 않습니다. 세로 방향에서는 작은 치수가 너비로 사용되고 가로 방향에서는 높이가 너비로 사용됩니다. 즉, `setSize`에 대한 다음 두 호출은 비디오 구성에 동일한 영향을 미칩니다.

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

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

네트워크 문제 처리

로컬 디바이스의 네트워크 연결이 끊어지면 SDK는 사용자 작업 없이 내부적으로 재연결을 시도합니다. SDK에서 재연결이 성공적이지 않아 사용자 작업이 필요한 경우도 있습니다. 네트워크 연결 끊김과 관련된 두 가지 주요 오류가 있습니다.

- 오류 코드 1400, 메시지: "PeerConnection 알 수 없는 네트워크 오류로 인해 손실되었습니다."
- 오류 코드 1300, 메시지: “재시도 횟수가 모두 소진됨”

첫 번째 오류가 수신되었지만 두 번째 오류는 수신되지 않은 경우 SDK가 여전히 스테이지에 연결되어 있으며 자동으로 연결 재설정을 시도합니다. 예방 조치로 전략 메서드의 반환 값을 변경하지 않고 `refreshStrategy`를 호출하여 수동 재연결 시도를 트리거할 수 있습니다.

두 번째 오류가 수신되면 SDK의 재연결 시도가 실패하고 로컬 디바이스가 더 이상 스테이지에 연결되지 않습니다. 이 경우 네트워크 연결이 다시 설정된 후 `join`을 호출하여 스테이지에 다시 참여해 보세요.

일반적으로 스테이지에 성공적으로 참가한 후 오류가 발생하면 SDK가 연결 재설정에 실패했음을 나타냅니다. 새 `Stage` 객체를 만들고 네트워크 상태가 개선되면 참가를 시도합니다.

Bluetooth 마이크 사용

Bluetooth 마이크 장치를 사용하여 게시하기 위해서는 Bluetooth SCO 연결을 시작해야 합니다.

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

알려진 문제 및 해결 방법

- Android 디바이스가 절전 모드로 전환되었다가 다시 작동하면 미리 보기가 멈춘 상태일 수 있습니다.

  해결 방법: 새 `Stage`를 생성하고 사용합니다.
- 참가자가 다른 참가자가 사용 중인 토큰으로 참가하면 별도의 오류 없이 첫 번째 연결이 끊어집니다.

  해결 방법: 없음
- 게시자가 게시하는 동안 간혹 구독자가 수신받는 게시 상태가 `inactive`인 경우가 발생할 수 있습니다.

  해결 방법: 세션에서 나간 다음 세션에 참가해 보세요. 문제가 계속되면 게시자를 위한 새 토큰을 생성하세요.
- 오디오 왜곡 문제는 스테이지 세션 중에 간헐적으로 발생할 수 있으며, 일반적으로 호출이 장시간 지속될 때 발생합니다.

  해결 방법: 오디오가 왜곡된 참가자는 세션을 나간 후 다시 참가하거나 오디오 게시를 취소하고 다시 게시함으로써 문제를 해결할 수 있습니다.

- 스테이지에 게시할 때는 외부 마이크가 지원되지 않습니다.

  해결 방법: 스테이지에 게시하기 위해 USB를 통해 연결된 외부 마이크를 사용하지 마세요.

- `createSystemCaptureSources`를 사용하여 화면을 공유하는 스테이지로 게시하는 것은 지원되지 않습니다.

  해결 방법: 사용자 지정 이미지 입력 소스 및 사용자 지정 오디오 입력 소스를 사용하여 시스템 캡처를 수동으로 관리합니다.

- `ImagePreviewView`가 상위에서 제거되면(예: `removeView()`가 상위에서 호출됨) `ImagePreviewView`가 즉시 해제됩니다. 다른 상위 뷰에 추가되면 `ImagePreviewView`에서 프레임을 표시하지 않습니다.

  해결 방법: `getPreview`를 사용하여 다른 미리 보기를 요청합니다.

- Android 12가 설치된 Samsung Galaxy S22/+를 사용하여 스테이지에 참가할 때 1401 오류가 발생하고 로컬 디바이스가 스테이지에 참가하지 못하거나 참가하지만 오디오가 재생되지 않을 수 있습니다.

  해결 방법: Android 13으로 업그레이드하세요.

- Android 13 기반 Nokia X20으로 스테이지에 참가하면 카메라가 열리지 않고 예외가 발생할 수 있습니다.

  해결 방법: 없음

- MediaTek Helio 칩셋이 장착된 장치는 원격 참가자의 비디오를 제대로 렌더링하지 못할 수 있습니다.

  해결 방법: 없음

- 일부 디바이스에서는 디바이스 OS가 SDK를 통해 선택한 것과 다른 마이크를 선택할 수 있습니다. 이는 Amazon IVS 브로드캐스트 SDK가 `VOICE_COMMUNICATION` 오디오 경로 정의 방법을 제어할 수 없기 때문이며, 오디오 경로가 디바이스 제조업체마다 다르기 때문입니다.

  해결 방법: 없음

- 일부 Android 비디오 인코더는 176x176 미만의 비디오 크기로 구성할 수 없습니다. 크기를 작게 구성하면 오류가 발생하고 스트리밍이 차단됩니다.

  해결 방법: 비디오 크기를 176x176 미만으로 구성하지 마십시오.

오류 처리

치명적인 오류와 치명적이지 않은 오류

오류 객체에는 BroadcastException의 "치명적" 부울 필드가 있습니다.

일반적으로 치명적인 오류는 스테이지 서버 연결과 관련이 있습니다(연결을 설정할 수 없거나 연결이 끊어져 복구할 수 없음). 애플리케이션은 스테이지를 다시 만들고 가능하면 새 토큰을 사용하거나 디바이스 연결이 복구되면 다시 참가해야 합니다.

치명적이지 않은 오류는 일반적으로 게시/구독 상태와 관련이 있으며 게시/구독 작업을 재시도하는 SDK에서 처리합니다.

이 속성을 확인할 수 있습니다.

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

참가 오류

잘못된 토큰

스테이지 토큰의 형식이 잘못된 경우 발생합니다.

SDK는 stage.join에 대한 호출에서 오류 코드 = 1000이고 fatal = true인 Java 예외를 발생시킵니다.

조치: 유효한 토큰을 생성한 후 다시 참가해 보세요.

만료된 토큰

스테이지 토큰이 만료된 경우 발생합니다.

SDK는 stage.join에 대한 호출에서 오류 코드 = 1001이고 fatal = true인 Java 예외를 발생시킵니다.

조치: 새 토큰을 생성한 후 다시 참가해 보세요.

유효하지 않거나 취소된 토큰

스테이지 토큰의 형식이 잘못되진 않았지만 스테이지 서버에서 거부된 경우 발생합니다. 이는 애플리케이션에서 제공하는 스테이지 렌더러를 통해 비동기적으로 보고됩니다.

SDK는 오류 코드 = 1026이고 fatal = true인 예외로 `onConnectionStateChanged`를 호출합니다.

조치: 유효한 토큰을 생성한 후 다시 참가해 보세요.

첫 참가 시 네트워크 오류

SDK가 스테이지 서버에 접속하여 연결을 설정할 수 없는 경우 발생합니다. 이는 애플리케이션에서 제공하는 스테이지 렌더러를 통해 비동기적으로 보고됩니다.

SDK는 오류 코드 = 1300이고 fatal = true인 예외로 `onConnectionStateChanged`를 호출합니다.

조치: 디바이스 연결이 복구될 때까지 기다린 후 다시 참가해 보세요.

이미 참가한 경우 네트워크 오류

디바이스의 네트워크 연결이 끊어지면 SDK와 스테이지 서버 연결이 끊어질 수 있습니다. 이는 애플리케이션에서 제공하는 스테이지 렌더러를 통해 비동기적으로 보고됩니다.

SDK는 오류 코드 = 1300이고 fatal = true인 예외로 `onConnectionStateChanged`를 호출합니다.

조치: 디바이스 연결이 복구될 때까지 기다린 후 다시 참가해 보세요.

게시/구독 오류

Initial

다음과 같은 여러 오류가 있습니다.

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

- SignallingSessionBadResponse (1203)

이는 애플리케이션에서 제공하는 스테이지 렌더러를 통해 비동기적으로 보고됩니다.

SDK는 제한된 횟수만큼 작업을 재시도합니다. 재시도 시 게시/구독 상태는 `ATTEMPTING_PUBLISH/ATTEMPTING_SUBSCRIBE`입니다. 재시도가 성공하면 상태가 `PUBLISHED/SUBSCRIBED`로 변경됩니다.

SDK는 관련 오류 코드와 fatal = false로 `onError`를 호출합니다.

조치: SDK가 자동으로 재시도하므로 조치가 필요하지 않습니다. 선택적으로 애플리케이션에서 전략을 새로 고쳐 추가 재시도를 강제할 수 있습니다.

이미 설정된 후 실패

게시 또는 구독이 설정된 후 실패할 수 있는데, 이는 대부분 네트워크 오류로 인한 것입니다. “네트워크 오류로 인해 피어 연결이 끊어짐”의 오류 코드는 1400입니다.

이는 애플리케이션에서 제공하는 스테이지 렌더러를 통해 비동기적으로 보고됩니다.

SDK는 게시/구독 작업을 재시도합니다. 재시도 시 게시/구독 상태는 `ATTEMPTING_PUBLISH/ATTEMPTING_SUBSCRIBE`입니다. 재시도가 성공하면 상태가 `PUBLISHED/SUBSCRIBED`로 변경됩니다.

SDK는 오류 코드 = 1400이고 fatal = false인 `onError`를 호출합니다.

조치: SDK가 자동으로 재시도하므로 조치가 필요하지 않습니다. 선택적으로 애플리케이션에서 전략을 새로 고쳐 추가 재시도를 강제할 수 있습니다. 전체 연결이 끊어지는 경우 스테이지에 대한 연결도 실패할 가능성이 높습니다.

IVS 브로드캐스트 SDK: iOS 가이드(실시간 스트리밍)

IVS 실시간 스트리밍 iOS 브로드캐스트 SDK를 사용하면 참가자가 iOS에서 비디오를 전송하고 수신할 수 있습니다.

`AmazonIVSBroadcast` 모듈은 본 문서에서 설명하는 인터페이스를 구현합니다. 지원되는 작업은 다음과 같습니다.

- 스테이지 참가

- 스테이지의 다른 참가자에게 미디어 게시
- 스테이지에 있는 다른 참가자의 미디어 구독
- 스테이지에 게시된 비디오 및 오디오 관리 및 모니터링
- 각 피어 연결에 대한 WebRTC 통계 가져오기
- IVS 지연 시간이 짧은 스트리밍 iOS 브로드캐스트 SDK의 모든 작업

iOS 브로드캐스트 SDK의 최신 버전: 1.14.1 (릴리스 노트)

참조 문서: Amazon IVS iOS 브로드캐스트 SDK에서 사용할 수 있는 가장 중요한 방법에 대한 자세한 내용은 https://aws.github.io/ amazon-ivs-broadcast-docs /1.14.1/ios/의 참조 설명서를 참조하십시오.

샘플 코드: iOS 샘플 리포지토리 https://github.com/aws-samples/ GitHub amazon-ivs-broadcast-ios - sample을 참조하십시오.

플랫폼 요구 사항: iOS 14 이상

시작하기

라이브러리 설치

를 통해 브로드캐스트 SDK를 통합하는 것이 좋습니다. CocoaPods (또는 프레임워크를 프로젝트에 수동으로 추가할 수 있습니다.)

권장 사항: 브로드캐스트 SDK 통합 () CocoaPods

실시간 기능은 iOS 저지연 스트리밍 브로드캐스트 SDK(iOS Low-Latency Streaming broadcast SDK)의 하위 사양으로 게시됩니다. 이를 통해 고객은 기능 요구 사항에 따라 해당 기능을 포함 또는 제외할 수 있으며, 포함할 경우 패키지 크기가 증가합니다. 포함하면 패키지 크기가 커집니다.

릴리스는 라는 CocoaPods 이름으로 `AmazonIVSBroadcast` 게시됩니다. 이 종속성을 Podfile에 추가합니다.

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

이후 `pod install`을 실행하면 `.xcworkspace`에서 SDK를 사용할 수 있습니다.

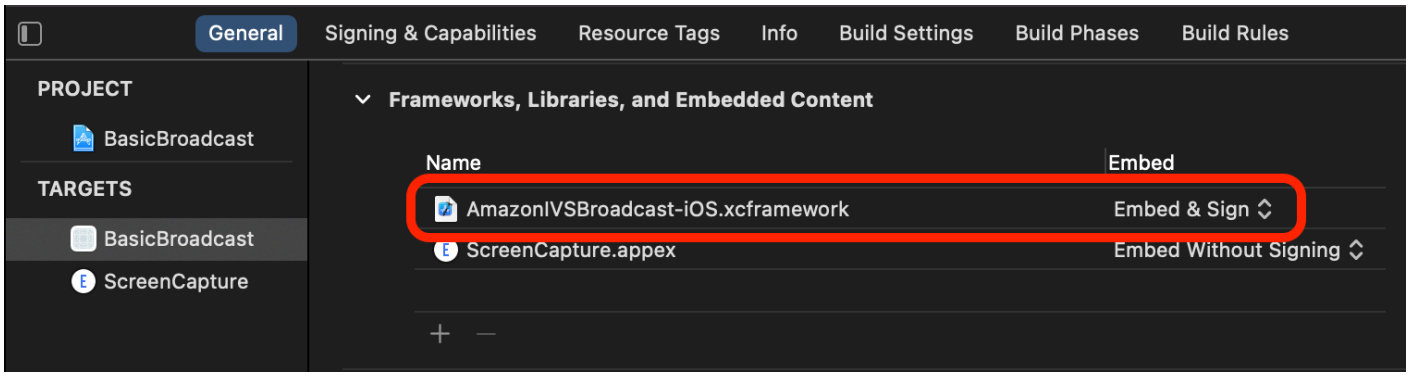
중요: IVS 실시간 스트리밍 브로드캐스트 SDK(스테이지 하위 사양 포함)에는 iOS 저지연 스트리밍 브로드캐스트 SDK의 모든 기능이 포함되어 있습니다. 두 SDK를 동일한 프로젝트에 통합하는 것은 불가

능합니다. Stage subspec via를 프로젝트에 CocoaPods 추가하는 경우, 포함된 Podfile에서 다른 줄은 모두 제거해야 합니다. `AmazonIVSBroadcast` 예를 들어, Podfile에 다음 두 줄이 모두 있으면 안 됩니다.

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

대체 방법: 수동으로 프레임워크 설치

1. https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip 에서 최신 버전을 다운로드하십시오.
2. 아카이브 콘텐츠의 압축을 풉니다. `AmazonIVSBroadcast.xcframework`에는 디바이스와 시뮬레이터 모두에 대한 SDK가 포함되어 있습니다.
3. 애플리케이션 대상에 대해 일반 탭의 프레임워크, 라이브러리 및 포함된 콘텐츠 섹션으로 끌어 `AmazonIVSBroadcast.xcframework`를 포함합니다.



권한 요청

앱에서 사용자의 카메라 및 마이크에 액세스할 수 있는 권한을 요청해야 합니다. (이는 Amazon IVS에만 국한되지 않으며 카메라와 마이크에 액세스해야 하는 모든 애플리케이션에 필요합니다.)

사용자가 부여된 권한이 이미 있는지 확인하고 없을 시 권한을 요청합니다.

```
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.
}
```

카메라와 마이크에 각각 액세스하려는 경우 `.video` 및 `.audio` 미디어 유형 모두에 위와 같이 권한을 요청해야 합니다.

또한 `NSCameraUsageDescription` 및 `NSMicrophoneUsageDescription`에 대한 항목을 `Info.plist`에 추가해야 합니다. 추가하지 않을 경우 권한 요청시 앱이 중단됩니다.

애플리케이션 유휴 타이머 사용 중지

이 단계는 선택 사항이지만 권장됩니다. 브로드캐스트 SDK를 사용하는 동안 디바이스가 절전 모드로 전환되는 것을 방지함으로써 브로드캐스트가 중단되는 것을 막습니다.

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

게시 및 구독

개념

실시간 기능의 3가지 핵심 개념은 스테이지, 전략 및 렌더러입니다. 설계 목표는 작동하는 제품을 구축하는 데 필요한 클라이언트 측 로직의 수를 최소화하는 것입니다.

단계

`IVSStage` 클래스는 호스트 애플리케이션과 SDK 사이의 주요 상호 작용 지점입니다. 클래스는 스테이지 자체를 나타내며 스테이지에 참가하고 나가는 데 사용됩니다. 스테이지를 만들거나 참가하려면 제어 플레인에서 유효하고 만료되지 않은 토큰 문자열(`token`으로 표시됨)이 필요합니다. 스테이지 참가 및 나가기는 간단합니다.

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

try stage.join()
```

```
stage.leave()
```

IVSStage 클래스는 IVSStageRenderer 및 IVSErrorDelegate가 첨부될 수 있는 위치이기도 합니다.

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

Strategy

IVSStageStrategy 프로토콜은 호스트 애플리케이션이 원하는 스테이지 상태를 SDK에 전달하는 방법을 제공합니다. shouldSubscribeToParticipant, shouldPublishParticipant 및 streamsToPublishForParticipant 함수를 구현해야 합니다. 모든 함수를 아래에서 설명합니다.

참가자 구독

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

원격 참가자가 스테이지에 참가하면 SDK는 호스트 애플리케이션에 해당 참가자의 원하는 구독 상태를 쿼리합니다. 옵션은 .none, .audioOnly 및 .audioVideo입니다. 이 함수의 값을 반환할 때 호스트 애플리케이션은 게시 상태, 현재 구독 상태 또는 스테이지 연결 상태에 대해 걱정할 필요가 없습니다. .audioVideo가 반환되는 경우 SDK는 구독 전에 원격 참가자가 게시할 때까지 기다리고, 프로세스 전반에 걸쳐 렌더러를 통해 호스트 애플리케이션을 업데이트합니다.

다음은 샘플 구현입니다.

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

항상 모든 참가자가 서로 보기를 원하는 호스트 애플리케이션(예: 비디오 채팅 애플리케이션)을 위한 이 기능의 전체 구현입니다.

고급 구현도 가능합니다. IVSParticipantInfo의 attributes 속성을 사용하여 서버에서 제공하는 특성을 기반으로 참가자를 선택적으로 구독합니다.

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

이를 통해 중재자가 직접 보거나 듣지 않고 모든 게스트를 모니터링할 수 있는 스테이지를 만들 수 있습니다. 호스트 애플리케이션은 추가 비즈니스 로직을 사용하여 중재자가 서로를 볼 수는 있지만 게스트에게는 보이지 않도록 할 수 있습니다.

게시

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

스테이지에 연결되면 SDK는 호스트 애플리케이션을 쿼리하여 특정 참가자가 게시해야 하는지 여부를 확인합니다. 이는 제공된 토큰을 기반으로 게시할 권한이 있는 로컬 참가자에게만 호출됩니다.

다음은 샘플 구현입니다.

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

이는 사용자가 항상 게시하기를 원하는 표준 비디오 채팅 애플리케이션에 대한 구현입니다. 오디오 및 비디오를 음소거 또는 음소거 해제하여 즉시 숨기거나 보기/듣기가 가능하도록 할 수 있습니다. (게시/게시 취소를 사용할 수도 있지만 속도가 훨씬 느립니다. 음소거/음소거 해제는 가시성을 자주 변경하는 것이 바람직한 사용 사례에 적합합니다.)

게시할 스트림 선택

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

게시할 때 이를 사용하여 게시해야 하는 오디오 및 비디오 스트림을 결정합니다. 이에 대해서는 미디어 스트림 게시에서 자세히 설명합니다.

전략 업데이트

전략은 동적이어야 합니다. 위 함수 중에서 반환되는 값은 언제든지 변경될 수 있습니다. 예를 들어 호스트 애플리케이션이 최종 사용자가 버튼을 탭할 때까지 게시하지 않으려는 경우, shouldPublishParticipant에서 변수를 반환할 수 있습니다(예: hasUserTappedPublishButton). 최종 사용자의 상호 작용에 따라 변수가 변경되면 stage.refreshStrategy()를 호출하여 SDK에 최신 값에 대한 전략을 쿼리하고 변경된 사항만 적용하도록 신호를 보냅니다. SDK에서 shouldPublishParticipant 값이 변경된 것을 관찰하면 게시 프로세스가 시작됩니다. SDK 쿼리와 모든 함수가 이전과 동일한 값을 반환하는 경우 refreshStrategy 호출 시 스테이지가 수정되지 않습니다.

shouldSubscribeToParticipant의 반환 값이 .audioVideo에서 .audioOnly로 변경된 경우 이전에 비디오 스트림이 존재했다면 반환된 값이 변경된 모든 참가자에 대한 비디오 스트림이 제거됩니다.

일반적으로 스테이지는 전략을 사용하여 이전 전략과 현재 전략 간의 차이를 가장 효율적으로 적용하므로 호스트 애플리케이션에서 이를 올바르게 관리하는 데 필요한 모든 상태에 대해 걱정할 필요가 없습니다. 이로 인해 stage.refreshStrategy() 호출은 전략이 변경되지 않는 한 아무 소용도 없으므로 소모량이 적은 작업이라고 생각하면 됩니다.

렌더러

IVSStageRenderer 프로토콜은 호스트 애플리케이션에 스테이지 상태를 전달하는 방법을 제공합니다. 호스트 애플리케이션의 UI 업데이트는 대체로 렌더러에서 제공하는 이벤트에 의해 전적으로 이루어질 수 있습니다. 렌더러는 다음과 같은 함수를 제공합니다.

```
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?)
```

렌더러에서 제공하는 정보가 전략의 반환 값에 영향을 미칠 것으로 예상되지는 않습니다. 예를 들어, `shouldSubscribeToParticipant`의 반환 값은 `participant:didChangePublishState`가 호출될 때 변경되지 않을 것으로 예상됩니다. 호스트 애플리케이션이 특정 참가자를 구독하려는 경우 해당 참가자의 게시 상태와 무관하게 원하는 구독 유형을 반환해야 합니다. SDK는 원하는 전략 상태가 스테이지 상태를 기반을 정확한 시간에 실행되도록 하는 역할을 합니다.

게시 참가자만 `participantDidJoin`를 트리거하고, 참가자가 게시를 중단하거나 스테이지 세션에서 나갈 때마다 `participantDidLeave`가 트리거됩니다.

미디어 스트림 게시

내장 마이크 및 카메라와 같은 로컬 디바이스는 `IVSDeviceDiscovery`를 통해 검색됩니다. 다음은 전면 카메라와 기본 마이크를 선택한 다음 SDK에 게시될 `IVSLocalStageStreams`로 반환하는 예제입니다.

```
let devices = IVSDeviceDiscovery.listLocalDevices()

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

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

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

참가자 표시 및 제거

구독이 완료되면 렌더러의 didAddStreams 함수를 통해 IVSStageStream 객체 배열을 받게 됩니다. 이 참가자에 대한 오디오 수준 통계를 미리 보거나 수신하려면 스트림에서 기본IVSDevice 객체에 액세스할 수 있습니다.

```
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 */
    })
}
```

참가자가 게시를 중단하거나 참가자 구독이 취소되면 제거된 스트림과 함께 didRemoveStreams 함수가 호출됩니다. 호스트 애플리케이션은 이를 신호로 사용하여 보기 계층 구조에서 참가자의 비디오 스트림을 제거해야 합니다.

didRemoveStreams는 다음을 포함하여 스트림이 제거될 수 있는 모든 시나리오에서 호출됩니다.

- 원격 참가자가 게시를 중단합니다.
- 로컬 디바이스가 구독을 취소하거나 구독을 .audioVideo에서 .audioOnly로 변경합니다.
- 원격 참가자가 스테이지를 나갑니다.
- 로컬 참가자가 스테이지를 나갑니다.

모든 시나리오에서 didRemoveStreams가 호출되므로 원격 또는 로컬 나가기 작업 중에 UI에서 참가자를 제거하는 사용자 지정 비즈니스 로직이 필요하지 않습니다.

미디어 스트림 음소거 및 음소거 해제

IVSLocalStageStream 객체에는 스트림의 음소거 여부를 제어하는 setMuted 함수가 있습니다. 이 함수는 streamsToPublishForParticipant 전략 함수에서 반환되기 전이나 후에 스트림에서 호출할 수 있습니다.

중요: refreshStrategy 호출 이후 streamsToPublishForParticipant에 의해 새 IVSLocalStageStream 객체 인스턴스가 반환되면 새 스트림 객체의 음소거 상태가 스테이지에 적용됩니다. 새 IVSLocalStageStream 인스턴스를 만들 때는 예상되는 음소거 상태가 유지되도록 주의해야 합니다.

원격 참가자 미디어 음소거 상태 모니터링

참가자가 비디오 또는 오디오 스트림의 음소거 상태를 변경할 때 변경된 스트림 배열과 함께 렌더러 didChangeMutedStreams 함수가 호출됩니다. IVSStageStream의 isMuted 속성을 사용하여 UI를 적절히 업데이트합니다.

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

스테이지 구성 생성

스테이지의 비디오 구성 값을 사용자 지정하려면 IVSLocalStageStreamVideoConfiguration을 사용합니다.

```
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
```

WebRTC 통계 가져오기

게시 스트림 또는 구독 스트림에 대한 최신 WebRTC 통계를 가져오려면 IVSStageStream에서 requestRTCStats를 사용합니다. 수집이 완료되면 IVSStageStream에서 설정할 수 있는 IVSStageStreamDelegate를 통해 통계를 받습니다. WebRTC 통계를 지속적으로 수집하려면 Timer에서 이 함수를 호출합니다.

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

참가자 특성 가져오기

CreateParticipantToken 엔드포인트 요청에서 특성을 지정하는 경우 IVSParticipantInfo 속성에서 특성을 볼 수 있습니다.

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

백그라운드에서 세션 계속하기

앱이 백그라운드로 전환될 때 원격 오디오를 들으면서 스테이지에 계속 있을 수 있지만, 본인이 이미지와 오디오를 계속 전송할 수는 없습니다. IVSStrategy 구현을 업데이트하여 게시를 중단하고 .audioOnly를 구독해야 합니다(또는 해당하는 경우 .none).

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

그런 다음 stage.refreshStrategy()를 호출합니다.

동시 방송을 사용한 계층화된 인코딩 활성화/비활성화

미디어 스트림을 게시할 때 SDK는 고품질 및 저품질 비디오 스트림을 전송하므로 원격 참가자는 다운링크 대역폭이 제한되어 있어도 스트림을 구독할 수 있습니다. 동시 방송을 사용한 계층화된 인코딩이 기본적으로 켜져 있습니다. IVSSimulcastConfiguration으로 이를 비활성화할 수 있습니다.

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

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

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

IVS 채널로 스테이지 브로드캐스트

스테이지를 브로드캐스트하려면 별도의 IVSBroadcastSession을 만든 다음 위에서 설명한 대로 SDK로 브로드캐스트하기 위한 일반적인 지침을 따릅니다. IVSStageStream의 device 속성은 위 코드 조각에 표시된 대로 IVSImageDevice 또는 IVSAudioDevice입니다. IVSBroadcastSession.mixer에 연결하여 전체 스테이지를 사용자 지정 가능한 레이아웃으로 브로드캐스트할 수 있습니다.

필요에 따라 스테이지를 합성하고 IVS 지연 시간이 짧은 채널로 브로드캐스트하여 더 많은 청중에게 다가갈 수 있습니다. IVS 지연 시간이 짧은 스트리밍 사용 설명서의 Amazon IVS 스트림에서 여러 호스트 활성화를 참조하세요.

iOS에서 카메라 해상도와 프레임 속도를 선택하는 방식

브로드캐스트 SDK로 관리되는 카메라는 해상도와 프레임 속도 (frames-per-second또는 FPS) 를 최적화하여 열 발생과 에너지 소비를 최소화합니다. 이 섹션에서는 호스트 애플리케이션을 사용 사례에 따라 최적화하는 데 도움이 되도록 해상도와 프레임 속도를 선택하는 방법을 설명합니다.

IVSCamera로 IVSLocalStageStream을 생성하면 카메라가 IVSLocalStageStreamVideoConfiguration.targetFramerate의 프레임 속도와 IVSLocalStageStreamVideoConfiguration.size의 해상도에 따라 최적화됩니다. IVSLocalStageStream.setConfiguration을 호출하면 카메라가 더 새로운 값으로 업데이트됩니다.

카메라 미리 보기

IVSCamera를 IVSBroadcastSession 또는 IVSStage에 연결하지 않고 미리 보기를 생성하는 경우 기본값은 해상도 1080p, 프레임 속도 60fps입니다.

스테이지 브로드캐스팅

IVSBroadcastSession을 사용하여 IVSStage를 브로드캐스트하면 SDK에서는 양 세션의 기준을 충족하는 해상도와 프레임 속도로 카메라 최적화를 시도합니다.

예를 들어, 브로드캐스트의 프레임 속도가 15FPS, 해상도가 1080p로 설정되어 있고 스테이지의 프레임 속도가 30FPS, 해상도가 720p인 경우 SDK에서는 카메라 구성을 프레임 속도를 30FPS, 해상

도를 1080p로 선택합니다. IVSBroadcastSession에서는 다른 모든 프레임을 카메라에서 삭제하고, IVSStage에서는 1080p 이미지를 720p로 축소합니다.

호스트 애플리케이션에서 IVSBroadcastSession과 IVSStage 모두를 카메라와 함께 사용할 계획이라면 각 구성의 targetFramerate 속성과 size 속성이 일치하는 것이 좋습니다. 일치하지 않으면 비디오를 캡처하는 동안 카메라가 자체적으로 재구성되느라 비디오-샘플 전송이 잠시 지연될 수 있습니다.

동일한 값으로 설정했을 때 호스트 애플리케이션의 사용 사례가 충족되지 않는 경우, 품질이 더 높은 카메라를 먼저 생성하면 품질이 더 낮은 세션이 추가될 때 카메라가 자체적으로 재구성되지 않도록 할 수 있습니다. 예를 들어, 1080p 및 30FPS로 브로드캐스트한 다음 720p 및 30FPS로 설정된 스테이지를 조인하면 카메라가 자체적으로 재구성되지 않으며 비디오가 중단되지 않고 계속됩니다. 이는 720p가 1080p 이하이고 30FPS가 30FPS 이하이기 때문입니다.

임의 프레임 속도, 해상도 및 종횡비

대다수 카메라 하드웨어는 30FPS의 720p 또는 60FPS의 1080p와 같은 일반적인 형식을 정확히 일치시킬 수 있습니다. 그러나 모든 형식을 정확히 일치시킬 수는 없습니다. 브로드캐스트 SDK에서는 다음과 같은 규칙(우선순위 오름차순)에 따라 카메라 구성을 선택합니다.

1. 해상도의 너비와 높이는 원하는 해상도 이상이지만, 이 제약 조건 내에서의 가장 작은 값입니다.
2. 프레임 속도는 원하는 프레임 속도 이상이지만, 이 제약 조건 내에서의 가장 작은 값입니다.
3. 종횡비는 원하는 종횡비와 일치합니다.
4. 일치하는 형식이 여러 개인 경우 시야가 가장 큰 형식이 사용됩니다.

다음은 두 가지 예제입니다.

- 호스트 애플리케이션에서 120FPS의 4k로 브로드캐스트를 시도하고 있습니다. 선택한 카메라에서는 60FPS의 4k 또는 120FPS의 1080p만 지원합니다. 프레임 속도 규칙보다 해상도 규칙의 우선순위가 높기 때문에 선택한 형식이 60FPS의 4k가 됩니다.
- 1910x1070이라는 불규칙한 해상도가 요청됩니다. 카메라에서는 1920x1080을 사용합니다. 주의: 1921x1080과 같은 해상도를 선택하면 카메라가 사용 가능한 다음 해상도(예: 2592x1944)로 스케일업되어 CPU 및 메모리-대역폭 페널티가 발생합니다.

Android는 어떤가요?

Android에서는 iOS처럼 해상도나 프레임 속도가 즉시 조정되지 않으므로 Android 브로드캐스트 SDK는 영향을 받지 않습니다.

알려진 문제 및 해결 방법

- Bluetooth 오디오 경로를 변경하면 예기치 않은 결과가 발생할 수 있습니다. 세션 중 새로운 디바이스를 연결하면 iOS가 입력 경로를 자동으로 변경할 수도 또는 변경을 하지 못할 수도 있습니다. 또한 연결된 여러 Bluetooth 헤드셋을 동시에 선택할 수 없습니다. 이는 일반 브로드캐스트 및 스테이지 세션 모두에서 발생합니다.

  해결 방법: Bluetooth 헤드셋을 사용하려는 경우 브로드캐스트 또는 스테이지를 시작하기 전에 헤드셋을 연결하고 세션 전체에서 연결된 상태로 둡니다.
- iPhone 14, iPhone 14 Plus, iPhone 14 Pro 또는 iPhone 14 Pro Max를 사용하는 참가자는 다른 참가자에게 오디오 에코 문제를 일으킬 수 있습니다.

  해결 방법: 영향을 받는 디바이스를 사용하는 참가자는 헤드폰을 사용하여 다른 참가자의 에코 문제를 방지할 수 있습니다.
- 참가자가 다른 참가자가 사용 중인 토큰으로 참가하면 별도의 오류 없이 첫 번째 연결이 끊어집니다.

  해결 방법: 없음
- 게시자가 게시하는 동안 간혹 구독자가 수신받는 게시 상태가 `inactive`인 경우가 발생할 수 있습니다.

  해결 방법: 세션에서 나간 다음 세션에 참가해 보세요. 문제가 계속되면 게시자를 위한 새 토큰을 생성하세요.
- 참가자가 게시 또는 구독 중일 때 네트워크가 안정적인 경우에도 네트워크 문제로 인한 연결 해제를 나타내는 코드 1400과 함께 오류가 발생할 수 있습니다.

  해결 방법: 다시 게시하거나 구독해 보세요.
- 오디오 왜곡 문제는 스테이지 세션 중에 간헐적으로 발생할 수 있으며, 일반적으로 호출이 장시간 지속될 때 발생합니다.

  해결 방법: 오디오가 왜곡된 참가자는 세션을 나간 후 다시 참가하거나 오디오 게시를 취소하고 다시 게시함으로써 문제를 해결할 수 있습니다.

오류 처리

치명적인 오류와 치명적이지 않은 오류

오류 객체에는 “치명적” 부울 필드가 있습니다. 이는 부울을 포함하는 `IVSBroadcastErrorIsFatalKey`의 딕셔너리 항목입니다.

일반적으로 치명적인 오류는 스테이지 서버 연결과 관련이 있습니다(연결을 설정할 수 없거나 연결이 끊어져 복구할 수 없음). 애플리케이션은 스테이지를 다시 만들고 가능하면 새 토큰을 사용하거나 디바이스 연결이 복구되면 다시 참가해야 합니다.

치명적이지 않은 오류는 일반적으로 게시/구독 상태와 관련이 있으며 게시/구독 작업을 재시도하는 SDK에서 처리합니다.

이 속성을 확인할 수 있습니다.

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

참가 오류

잘못된 토큰

스테이지 토큰의 형식이 잘못된 경우 발생합니다.

SDK에서 오류 코드가 1000이고 IVS가 '예'인 Swift 예외가 발생합니다. BroadcastErrorIsFatalKey

조치: 유효한 토큰을 생성한 후 다시 참가해 보세요.

만료된 토큰

스테이지 토큰이 만료된 경우 발생합니다.

SDK에서 오류 코드가 1001이고 IVS가 '예'인 Swift 예외가 발생합니다. BroadcastErrorIsFatalKey

조치: 새 토큰을 생성한 후 다시 참가해 보세요.

유효하지 않거나 취소된 토큰

스테이지 토큰의 형식이 잘못되진 않았지만 스테이지 서버에서 거부된 경우 발생합니다. 이는 애플리케이션에서 제공하는 스테이지 렌더러를 통해 비동기적으로 보고됩니다.

SDK 호출은 오류 코드가 1026이고 `stage(didChange connectionState, withError error)` IVS는 '예'입니다. BroadcastErrorIsFatalKey

조치: 유효한 토큰을 생성한 후 다시 참가해 보세요.

첫 참가 시 네트워크 오류

SDK가 스테이지 서버에 접속하여 연결을 설정할 수 없는 경우 발생합니다. 이는 애플리케이션에서 제공하는 스테이지 렌더러를 통해 비동기적으로 보고됩니다.

SDK `stage(didChange connectionState, withError error)` 호출은 오류 코드가 1300이고 IVS는 '예'입니다. BroadcastErrorIsFatalKey

조치: 디바이스 연결이 복구될 때까지 기다린 후 다시 참가해 보세요.

이미 참가한 경우 네트워크 오류

디바이스의 네트워크 연결이 끊어지면 SDK와 스테이지 서버 연결이 끊어질 수 있습니다. 이는 애플리케이션에서 제공하는 스테이지 렌더러를 통해 비동기적으로 보고됩니다.

SDK `stage(didChange connectionState, withError error)` 호출은 오류 코드가 1300이고 IVS 값이 예인 것으로 표시됩니다. BroadcastErrorIsFatalKey

조치: 디바이스 연결이 복구될 때까지 기다린 후 다시 참가해 보세요.

게시/구독 오류

Initial

다음과 같은 여러 오류가 있습니다.

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

이는 애플리케이션에서 제공하는 스테이지 렌더러를 통해 비동기적으로 보고됩니다.

SDK는 제한된 횟수만큼 작업을 재시도합니다. 재시도 시 게시/구독 상태는 `ATTEMPTING_PUBLISH/ATTEMPTING_SUBSCRIBE`입니다. 재시도가 성공하면 상태가 `PUBLISHED/SUBSCRIBED`로 변경됩니다.

SDK 호출은 관련 오류 코드와 `IVSErrorSourceDelegate:didEmitError` 함께 호출되며 IVS는 BroadcastErrorIsFatalKey 아니오입니다.

조치: SDK가 자동으로 재시도하므로 조치가 필요하지 않습니다. 선택적으로 애플리케이션에서 전략을 새로 고쳐 추가 재시도를 강제할 수 있습니다.

이미 설정된 후 실패

게시 또는 구독이 설정된 후 실패할 수 있는데, 이는 대부분 네트워크 오류로 인한 것입니다. “네트워크 오류로 인해 피어 연결이 끊어짐”의 오류 코드는 1400입니다.

이는 애플리케이션에서 제공하는 스테이지 렌더러를 통해 비동기적으로 보고됩니다.

SDK는 게시/구독 작업을 재시도합니다. 재시도 시 게시/구독 상태는 `ATTEMPTING_PUBLISH/ATTEMPTING_SUBSCRIBE`입니다. 재시도가 성공하면 상태가 `PUBLISHED/SUBSCRIBED`로 변경됩니다.

SDK `didEmitError` 호출은 오류 코드가 1400이고 IVS는 아니오입니다. BroadcastErrorIsFatalKey

조치: SDK가 자동으로 재시도하므로 조치가 필요하지 않습니다. 선택적으로 애플리케이션에서 전략을 새로 고쳐 추가 재시도를 강제할 수 있습니다. 전체 연결이 끊어지는 경우 스테이지에 대한 연결도 실패할 가능성이 높습니다.

IVS 브로드캐스트 SDK: 사용자 지정 이미지 소스(실시간 스트리밍)

사용자 지정 이미지 입력 소스를 사용하면 애플리케이션이 사전 설정된 카메라로 제한되는 대신 브로드캐스트 SDK에 자체 이미지 입력을 제공할 수 있습니다. 사용자 지정 이미지 소스는 반투명 워터마크나 '잠시 기다려 주세요' 같은 정적 이미지처럼 간단한 이미지일 수도 있고, 카메라에 뷰티 필터를 추가하는 등 앱에서 추가 사용자 지정을 허용할 수도 있습니다.

카메라를 사용자 지정 제어하기 위해 사용자 지정 이미지 입력 소스를 사용하는 경우(예: 카메라 액세스가 필요한 뷰티 필터 라이브러리 사용) 브로드캐스트 SDK는 더이상 카메라 관리 책임이 없습니다. 대신에 카메라의 수명 주기를 올바르게 관리할 책임은 애플리케이션에 부여됩니다. 애플리케이션이 카메라를 관리하는 방법에 대한 공식 플랫폼 문서를 참조하세요.

Android

`DeviceDiscovery` 세션을 생성한 후 이미지 입력 소스를 생성합니다.

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

이 메서드는 표준 Android Surface에서 지원하는 이미지 소스인 CustomImageSource을(를) 반환합니다. 하위 클래스 SurfaceSource는 크기를 조정하고 회전할 수 있습니다. 또한 ImagePreviewView을(를) 생성하여 콘텐츠의 미리 보기를 표시할 수 있습니다.

기본 Surface 검색 방법:

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

이 Surface은(는) Camera2, OpenGL ES 및 기타 라이브러리와 같은 이미지 제작자의 출력 버퍼로 사용할 수 있습니다. 가장 간단한 사용 사례는 정적 비트맵 또는 색상을 Surface의 캔버스에 직접 그리는 것입니다. 그러나 많은 라이브러리(뷰티 필터 라이브러리 등)는 애플리케이션이 렌더링을 위해 외부 Surface을(를) 지정할 수 있도록 하는 메서드를 제공합니다. 이러한 메서드를 사용하여 Surface을(를) 필터 라이브러리에 전달하여, 라이브러리가 브로드캐스트 세션에서 스트리밍할 수 있도록 처리된 프레임을 출력할 수 있습니다.

CustomImageSource는 LocalStageStream에 래핑되고 Stage에 게시하기 위해 StageStrategy에 의해 반환될 수 있습니다.

iOS

DeviceDiscovery 세션을 생성한 후 이미지 입력 소스를 생성합니다.

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

이 메서드는 애플리케이션이 CMSampleBuffers을(를) 수동으로 제출하도록 허용하는 이미지 소스인 IVSCustomImageSource을(를) 반환합니다. 지원되는 픽셀 형식은 iOS 브로드캐스트 SDK 참조를 참조하세요. 현재 버전에 대한 최신 링크는 최신 브로드캐스트 SDK 릴리스 Amazon IVS 릴리스 정보에 있습니다.

사용자 지정 소스에 제출된 샘플은 스테이지로 스트리밍됩니다.

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

스트리밍 비디오의 경우 콜백에서 이 메서드를 사용하세요. 예를 들어 카메라를 사용하는 경우 AVCaptureSession에서 새 샘플 버퍼를 받을 때마다 애플리케이션이 해당 샘플 버퍼를 사용자 정의 이미지 소스로 전달할 수 있습니다. 원하는 경우 애플리케이션은 샘플을 사용자 정의 이미지 소스에 제출하기 전에 추가 처리(뷰티 필터 등) 를 적용할 수 있습니다.

IVSCustomImageSource는 IVSLocalStageStream에 래핑되고 Stage에 게시하기 위해 IVSStageStrategy에 의해 반환될 수 있습니다.

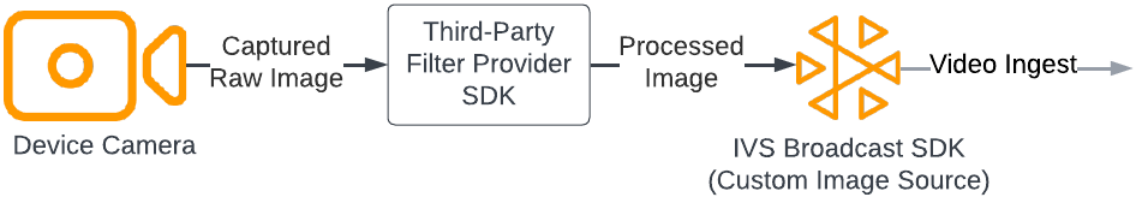
IVS 브로드캐스트 SDK: 타사 카메라 필터(실시간 스트리밍)

이 가이드에서는 사용자가 이미 사용자 지정 이미지 소스에 익숙하고 IVS 실시간 스트리밍 브로드캐스트 SDK를 애플리케이션에 통합하는 데 익숙하다고 가정합니다.

라이브 스트림 크리에이터는 카메라 필터를 사용하여 얼굴 또는 배경 모양을 확대하거나 대체할 수 있습니다. 이를 통해 잠재적으로 시청자 참여도를 높이고 시청자를 끌어들이며 라이브 스트리밍 환경을 개선할 수 있습니다.

타사 카메라 필터 통합

필터 SDK의 출력을 사용자 지정 이미지 입력 소스에 공급하여 타사 카메라 필터 SDK를 IVS 브로드캐스트 SDK와 통합할 수 있습니다. 사용자 지정 이미지 입력 소스를 사용하면 애플리케이션이 브로드캐스트 SDK에 자체 이미지 입력을 제공할 수 있습니다. 타사 필터 공급자의 SDK는 카메라의 수명 주기를 관리하여 카메라의 이미지를 처리하고, 필터 효과를 적용하며, 사용자 지정 이미지 소스에 전달할 수 있는 형식으로 출력할 수 있습니다.



필터 효과가 적용된 카메라 프레임을 사용자 지정 이미지 입력 소스에 전달할 수 있는 형식으로 변환하는 기본 제공 메서드는 타사 필터 공급자의 설명서를 참조합니다. 그 프로세스는 사용되는 IVS 브로드캐스트 SDK 버전에 따라 다릅니다.

- 웹 - 필터 공급자는 출력을 캔버스 요소로 렌더링할 수 있어야 합니다. 그런 다음 captureStream 메서드를 사용하여 캔버스 콘텐츠의 MediaStream을 반환할 수 있습니다. 그러면 MediaStream을 LocalStageStream의 인스턴스로 변환하여 스테이지에 게시할 수 있습니다.
- Android - 필터 공급자의 SDK는 IVS 브로드캐스트 SDK에서 제공하는 Android Surface로 프레임을 렌더링하거나 프레임을 비트맵으로 변환할 수 있습니다. 비트맵을 사용하는 경우 잠금을 해제하고 캔버스에 쓰면 사용자 지정 이미지 소스에서 제공하는 기본 Surface로 렌더링할 수 있습니다.
- iOS - 타사 필터 공급자의 SDK는 필터 효과가 CMSampleBuffer로 적용된 카메라 프레임을 제공해야 합니다. 카메라 이미지가 처리된 후 CMSampleBuffer를 최종 출력으로 얻는 방법에 대한 자세한 내용은 타사 필터 공급업체 SDK의 설명서를 참조하세요.

BytePlus

Android

BytePlus 효과 SDK 설치 및 설정

BytePlus Effects SDK의 설치, 초기화 및 설정 방법에 대한 자세한 내용은 BytePlus Android 액세스 가이드를 참조하세요.

사용자 지정 이미지 소스 설정

SDK를 초기화한 후 사용자 지정 이미지 입력 소스에 필터 효과를 적용한 카메라 프레임을 피드합니다. 그러려면 `DeviceDiscovery` 객체의 인스턴스를 만들고 사용자 지정 이미지 소스를 생성해야 합니다. 카메라의 사용자 지정 제어를 위해 사용자 지정 이미지 입력 소스를 사용하는 경우 브로드캐스트 SDK는 더 이상 카메라 관리를 담당하지 않습니다. 대신 애플리케이션은 카메라의 수명 주기를 올바르게 처리합니다.

Java

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

출력을 비트맵으로 변환 및 사용자 지정 이미지 입력 소스 제공

BytePlus Effect SDK에서 필터 효과가 적용된 카메라 프레임을 IVS 브로드캐스트 SDK로 직접 전달하려면 BytePlus Effects SDK의 텍스처 출력을 비트맵으로 변환하세요. 이미지가 처리되면 SDK에서 `onDrawFrame()` 메서드를 호출합니다. `onDrawFrame()` 메서드는 Android의 GLSurfaceView.Renderer 인터페이스의 공개 메서드입니다. BytePlus에서 제공하는 Android 샘플 앱에서 이 메서드가 모든 카메라 프레임에서 호출되어 텍스처를 출력합니다. 동시에 이 텍스처를 비트맵으로 변환하고 사용자 지정 이미지 입력 소스에 공급하는 로직으로 `onDrawFrame()` 메서드를 보완할 수 있습니다. 다음 코드 샘플에서 볼 수 있듯이 BytePlus SDK에서 제공하는 `transferTextureToBitmap` 메서드를 사용하여 이 변환을 수행하세요. 이 메서드는 다음 코드 샘플과 같이 BytePlus Effects SDK의 com.bytedance.labcv.core.util.ImageUtil 라이브러리가 제공합니다. 그런 다음 결과 비트맵을 Surface 캔버스에 기록하여 `CustomImageSource`의 기본 Android `Surface`로 렌더링할 수 있습니다. `onDrawFrame()`을 여러 번 연속해서 호출하면 비트맵 시퀀스를 생성하고, 결합하면 비디오 스트림을 생성합니다.

Java

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


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

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

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

DeepAR

Android

DeepAR SDK를 Android IVS 브로드캐스트 SDK와 통합하는 방법에 대한 자세한 내용은 DeepAR의 Android 통합 가이드를 참조하세요.

iOS

DeepAR SDK를 iOS IVS 브로드캐스트 SDK와 통합하는 방법에 대한 자세한 내용은 DeepAR의 iOS 통합 가이드를 참조하세요.

Snap

웹

이 섹션에서는 웹 브로드캐스트 SDK를 사용하여 비디오를 게시 및 구독하는 방법을 이미 잘 알고 있다고 가정합니다.

Snap의 Camera Kit SDK를 IVS 실시간 스트리밍 웹 브로드캐스트 SDK와 통합하려면 다음이 필요합니다.

1. Camera Kit SDK 및 Webpack을 설치합니다. (이 예에서는 Webpack을 번들러로 사용하지만 원하는 번들러를 사용할 수 있습니다.)
2. `index.html`을 생성합니다.
3. 설정 요소를 추가하세요.
4. 참가자를 표시하고 설정하세요.
5. 연결된 카메라와 마이크를 표시하세요.
6. Camera Kit 세션을 생성하세요.
7. Lens를 가져오고 적용하세요.
8. Camera Kit 세션에서 캔버스로 출력을 렌더링하세요.
9. Camera Kit에 렌더링용 미디어 소스를 제공하고 `LocalStageStream`을 게시하세요.
10. Webpack 구성 파일을 생성합니다.

이러한 각 단계는 아래에서 설명하고 있습니다.

Camera Kit SDK 및 Webpack 설치

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

index.html 생성

다음으로 HTML 표준 문안을 생성하고 웹 브로드캐스트 SDK를 스크립트 태그로 가져오세요. 다음 코드에서는 사용 중인 브로드캐스트 SDK 버전으로 `<SDK version>`을 바꿔야 합니다.

JavaScript

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

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

  <title>Amazon IVS Real-Time Streaming Web Sample (HTML and JavaScript)</title>
```

```
  <!-- Fonts and Styling -->
  <link rel="stylesheet" href="https://fonts.googleapis.com/css?
family=Roboto:300,300italic,700,700italic" />
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/normalize/8.0.1/
normalize.css" />
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/milligram/1.4.1/
milligram.css" />
  <link rel="stylesheet" href="./index.css" />

  <!-- Stages in Broadcast SDK -->
  <script src="https://web-broadcast.live-video.net/<SDK version>/amazon-ivs-web-
broadcast.js"></script>
</head>

<body>
  <!-- Introduction -->
  <header>
    <h1>Amazon IVS Real-Time Streaming Web Sample (HTML and JavaScript)</h1>

    <p>This sample is used to demonstrate basic HTML / JS usage. <b><a href="https://
docs.aws.amazon.com/ivs/latest/userguide/multiple-hosts.html">Use the AWS CLI</
a></b> to create a <b>Stage</b> and a corresponding <b>ParticipantToken</b>.
 Multiple participants can load this page and put in their own tokens. You can <b><a
 href="https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-guides/stages#glossary"
 target="_blank">read more about stages in our public docs.</a></b></p>
  </header>
  <hr />

  <!-- Setup Controls -->

  <!-- Local Participant -->

  <hr style="margin-top: 5rem"/>

  <!-- Remote Participants -->

  <!-- Load all Desired Scripts -->

</body>

</html>
```

설정 요소 추가

카메라와 마이크를 선택하고 참가자 토큰을 지정하기 위한 HTML을 다음과 같이 생성하세요.

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>
```

그 아래에 HTML을 추가하여 로컬 및 원격 참가자의 카메라 피드를 표시하세요.

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>
```

카메라 설정을 위한 도우미 메서드와 번들 JavaScript 파일을 비롯한 추가 로직을 로드합니다. (이 섹션의 뒷부분에서는 Camera Kit를 모듈로 가져올 수 있도록 이러한 JavaScript 파일을 만들고 단일 파일로 번들링합니다. 번들 JavaScript 파일에는 Camera Kit를 설정하고, Lens를 적용하고, 스테이지에 Lens를 적용한 상태로 카메라 피드를 게시하기 위한 로직이 포함됩니다.)

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>
```

참가자 표시 및 설정

다음으로, 참가자를 표시하고 설정하는 데 사용할 도우미 메서드를 포함한 `helpers.js`를 생성하세요.

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;
}
```

연결된 카메라 및 마이크 표시

다음으로, 디바이스에 연결된 카메라와 마이크를 표시하는 도우미 메서드를 포함함 `media-devices.js`를 생성하세요.

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,
    },
  });
}
```

Camera Kit 세션 생성

카메라 피드에 Lens를 적용하고 스테이지에 피드를 게시하기 위한 로직이 포함된 `stages.js`를 생성합니다. 이 파일의 첫 번째 부분에서는 브로드캐스트 SDK와 Camera Kit 웹 SDK를 가져와서 각 SDK에 사용할 변수를 초기화합니다. Camera Kit 웹 SDK를 부트스트래핑한 후 `createSession`을 호출하여 Camera Kit 세션을 생성합니다. 캔버스 요소 객체는 세션으로 전달됩니다. 그러면 Camera Kit가 해당 캔버스로 렌더링하도록 지시합니다.

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 });
```

Lens 가져오기 및 적용

Lens를 가져오려면 Camera Kit 개발자 포털에서 찾을 수 있는 Lens 그룹 ID를 입력하세요. 이 예시에서는 반환되는 Lens 배열의 첫 번째 Lens를 적용하여 간소화했습니다.

JavaScript

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

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

Camera Kit 세션에서 캔버스로 출력 렌더링

captureStream 메서드를 사용하여 캔버스의 콘텐츠 MediaStream을 반환합니다. 캔버스는 Lens가 적용된 카메라 피드의 비디오 스트림을 포함합니다. 카메라와 마이크를 음소거하는 버튼의 이벤트 리스너와 스테이지 참여 및 퇴장을 위한 이벤트 리스너도 추가할 수 있습니다. 스테이지 참여를 위한 이벤트 리스너에서는 Camera Kit 세션을 전달하고 캔버스에서 가져온 MediaStream을 스테이지에 게시할 수 있도록 전달합니다.

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();
```

```
  });
};
```

Camera Kit에 렌더링용 미디어 소스 제공 및 LocalStageStream 게시

Lens가 적용된 비디오 스트림을 게시하려면 이전에 캔버스에서 캡처한 `setCameraKitSource`에서 전달하는 `MediaStream` 함수를 만드세요. 로컬 카메라 피드를 아직 통합하지 않았기 때문에 캔버스의 `MediaStream`은 현재 아무것도 하고 있지 않습니다. `getCamera` 도우미 메서드를 호출하고 `localCamera`에 할당하여 로컬 카메라 피드를 통합할 수 있습니다. 그런 다음 `localCamera`를 통해 로컬 카메라 피드와 세션 객체를 `setCameraKitSource`에 전달할 수 있습니다. `setCameraKitSource` 함수는 `createMediaStreamSource` 호출을 통해 로컬 카메라 피드를 CameraKit용 미디어 소스로 변환합니다. 그러면 `CameraKit`의 미디어 소스가 전면 카메라를 미러링하도록 변환됩니다. 그 다음 미디어 소스에 Lens 효과가 적용되고 `session.play()`를 호출하여 출력 캔버스에 렌더링됩니다.

이제 캔버스에서 캡처한 `MediaStream`에 Lens를 적용한 다음 스테이지에 게시할 수 있습니다. 이렇게 하려면 `MediaStream`에서 가져온 비디오 트랙을 사용하여 `LocalStageStream`을 만듭니다. 그러면 `LocalStageStream`의 인스턴스를 `StageStrategy`에 전달하여 게시할 수 있습니다.

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;
    },
  };
```

아래 나머지 코드는 스테이지를 만들고 관리하기 위한 코드입니다.

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();
```

Webpack 구성 파일 생성

`webpack.config.js`을 생성하고 다음 코드를 추가합니다. 위 로직을 번들링하고 가져오기 문으로 Camera Kit를 사용할 수 있습니다.

JavaScript

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

마지막으로 Webpack 구성 파일에 정의된 대로 JavaScript를 번들링하여 `npm run build`을 실행합니다. 그러면 웹 서버에서 HTML 및 JavaScript를 제공할 수 있습니다. 예를 들어, Python의 HTTP 서버를 사용하고 localhost:8000을 열어 결과를 확인할 수 있습니다.

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

Android

Snap의 Camera Kit SDK를 IVS Android 브로드캐스트 SDK와 통합하려면 Camera Kit SDK를 설치하고, Camera Kit 세션을 초기화하고, Lens를 적용하고, Camera Kit 세션의 출력을 사용자 지정 이미지 입력 소스에 공급해야 합니다.

Camera Kit SDK를 설치하려면 모듈의 `build.gradle` 파일에 다음을 추가하세요. `$cameraKitVersion`을 최신 Camera Kit SDK 버전으로 교체하세요.

Java

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

초기화하고 cameraKitSession을 가져오세요. 또한 Camera Kit는 Android의 CameraX API를 위한 편리한 래퍼를 제공하므로 Camera Kit로 CameraX를 사용하기 위해 복잡한 로직을 작성할 필요가 없습니다. CameraXImageProcessorSource 객체를 ImageProcessor의 소스로 사용할 수 있으며, 이를 통해 카메라 미리보기 스트리밍 프레임을 시작할 수 있습니다.

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();
   }
```

Lens 가져오기 및 적용

Camera Kit 개발자 포털의 Carousel에서 Lens를 구성하고 순서를 지정할 수 있습니다.

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);
      }));
```

```
});
```

브로드캐스트하려면 처리된 프레임을 사용자 지정 이미지 소스의 기본 Surface로 전송하세요. DeviceDiscovery 객체를 사용하고 CustomImageSource를 생성하여 SurfaceSource를 반환합니다. 그런 다음 CameraKit 세션의 출력을 SurfaceSource에서 제공하는 기본 Surface로 렌더링할 수 있습니다.

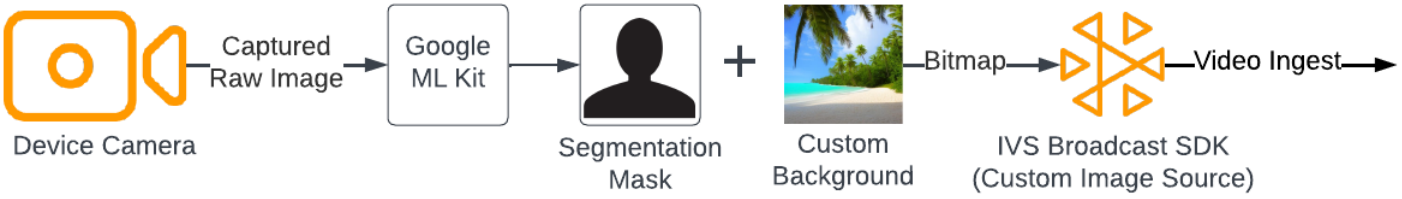
Java

```
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
```

배경 교체

배경 교체는 실시간 스트리밍 제작자가 배경을 변경할 수 있도록 하는 카메라 필터의 일종입니다. 다음 다이어그램에서 볼 수 있듯이 배경 교체 작업은 다음과 같습니다.

1. 라이브 카메라 피드에서 카메라 이미지를 가져옵니다.
2. Google ML Kit를 사용하여 전경 구성 요소와 배경 구성 요소로 구분합니다.
3. 생성된 분할 마스크를 사용자 지정 배경 이미지와 결합합니다.
4. 브로드캐스트용 사용자 지정 이미지 소스에 전달합니다.



웹

이 섹션에서는 웹 브로드캐스트 SDK를 사용하여 비디오를 게시 및 구독하는 방법을 이미 잘 알고 있다고 가정합니다.

라이브 스트림의 배경을 사용자 지정 이미지로 바꾸려면 MediaPipe 이미지 Segmenter와 셀카 분할 모델을 사용하세요. 이 기계 학습 모델은 비디오 프레임에서 전경 또는 배경에 있는 픽셀을 식별합니다. 그런 다음 비디오 피드의 전경 픽셀을 새 배경을 나타내는 사용자 지정 이미지에 복사하여 모델의 결과로 라이브 스트림의 배경을 바꿀 수 있습니다.

배경 교체를 IVS 실시간 스트리밍 웹 브로드캐스트 SDK와 통합하려면 다음이 작업을 수행해야 합니다.

1. MediaPipe와 Webpack을 설치하세요. (이 예에서는 Webpack을 번들러로 사용하지만 원하는 번들러를 사용할 수 있습니다.)
2. `index.html`을 생성합니다.
3. 미디어 요소를 추가하세요.
4. 스크립트 태그를 추가하세요.
5. `app.js`을 생성합니다.
6. 사용자 지정 배경 이미지를 불러오세요.
7. `ImageSegmenter`의 인스턴스를 만듭니다.
8. 캔버스로 비디오 피드를 렌더링하세요.
9. 배경 교체 로직을 생성하세요.
10. Webpack 구성 파일을 생성하세요.
11. JavaScript 파일을 번들링하세요.

MediaPipe 및 Webpack 설치

시작하려면 `@mediapipe/tasks-vision` 및 `webpack` npm 패키지를 설치하세요. 아래 예제에서는 Webpack을 JavaScript 번들러로 사용합니다. 원하는 경우 다른 번들러를 사용할 수 있습니다.

JavaScript

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

또한 webpack을 빌드 스크립트로 지정하도록 package.json을 업데이트해야 합니다.

JavaScript

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

index.html 생성

다음으로 HTML 표준 문안을 생성하고 웹 브로드캐스트 SDK를 스크립트 태그로 가져오세요. 다음 코드에서는 사용 중인 브로드캐스트 SDK 버전으로 <SDK version>을 바꿔야 합니다.

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>
```

미디어 요소 추가

다음으로 본문 태그에 비디오 요소 하나와 캔버스 요소 두 개를 추가합니다. 비디오 요소는 라이브 카메라 피드를 포함하며 MediaPipe 이미지 Segmenter에 대한 입력으로 사용합니다. 첫 번째 캔버스 요

소는 브로드캐스트될 피드의 미리보기를 렌더링하는 데 사용됩니다. 두 번째 캔버스 요소는 배경으로 사용할 사용자 지정 이미지를 렌더링하는 데 사용됩니다. 사용자 지정 이미지가 있는 두 번째 캔버스는 최종 캔버스에 프로그래밍 방식으로 픽셀을 복사하기 위한 소스로만 사용되므로 보기에서 숨겨집니다.

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>
```

스크립트 태그 추가

스크립트 태그를 추가하여 배경 교체 코드를 포함하는 번들 JavaScript 파일을 로드하고 스테이지에 게시합니다.

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

app.js 생성

다음으로 JavaScript 파일을 생성하여 HTML 페이지에서 만든 캔버스 및 비디오 요소의 요소 객체를 가져옵니다. `ImageSegmenter` 및 `FilesetResolver` 모듈을 가져옵니다. `ImageSegmenter` 모듈은 분할 작업을 수행하는 데 사용됩니다.

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";
```

다음으로, 사용자 카메라에서 MediaStream을 검색하는 `init()` 함수를 생성하고, 카메라 프레임 로드가 완료될 때마다 콜백 함수를 호출합니다. 스테이지에 참여 및 퇴장할 수 있는 버튼에 이벤트 리스너를 추가합니다.

스테이지에 참가할 때는 `segmentationStream`이라는 변수를 전달합니다. 캔버스 요소에서 캡처한 비디오 스트림에 배경을 나타내는 사용자 지정 이미지 위에 전경 이미지가 오버레이되어 있습니다. 나중에 이 사용자 지정 스트림을 사용하여 스테이지에 게시할 수 있는 `LocalStageStream`의 인스턴스를 생성할 수 있습니다.

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();
  });
};
```

사용자 지정 배경 이미지 로드

`init` 함수 아래쪽에 `initBackgroundCanvas`라는 함수를 호출하는 코드를 추가합니다. 이 함수는 로컬 파일에서 사용자 지정 이미지를 로드하여 캔버스에 렌더링합니다. 다음 단계에서 이 함수를 정의할 것입니다. 사용자 카메라에서 검색한 `MediaStream`을 비디오 객체에 할당합니다. 나중에 이 비디오 객체는 이미지 Segmenter로 전달됩니다. 또한 비디오 프레임 로드가 완료될 때마다 호출되도록 콜백 함수로서 이름이 `renderVideoToCanvas`인 함수를 설정하세요. 이후 단계에서 이 함수를 정의할 것입니다.

JavaScript

```
initBackgroundCanvas();

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

로컬 파일에서 이미지를 로드하는 `initBackgroundCanvas` 함수를 구현해 보겠습니다. 이 예에서는 해변 이미지를 사용자 지정 배경으로 사용합니다. 사용자 지정 이미지가 있는 캔버스는 카메라 피드를 포함한 캔버스 요소의 전경 픽셀과 병합되므로 디스플레이에서 숨겨집니다.

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);
  };
};
```

ImageSegmenter 인스턴스 생성

다음으로 이미지를 분할하고 결과를 마스크로 반환하는 `ImageSegmenter` 인스턴스를 생성합니다. `ImageSegmenter`의 인스턴스를 생성할 때는 셀카 분할 모델을 사용하게 됩니다.

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,
  });
};
```

캔버스로 비디오 피드 렌더링

다음으로, 비디오 피드를 다른 캔버스 요소에 렌더링하는 함수를 생성합니다. Canvas 2D API를 사용하여 전경 픽셀을 추출하려면 비디오 피드를 캔버스로 렌더링해야 합니다. 이 작업을 수행하는 동안 segmentforVideo 메서드를 사용하여 비디오 프레임의 배경에서 전경을 구분하여 비디오 프레임을 `ImageSegmenter` 인스턴스로 전달합니다. segmentforVideo 메서드가 반환되면 사용자 지정 콜백 함수 `replaceBackground`를 호출하여 배경 교체를 수행합니다.

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);
};
```

배경 교체 로직 생성

사용자 지정 배경 이미지를 카메라 피드의 전경과 병합하여 배경을 대체하는 `replaceBackground` 함수를 생성하세요. 함수는 먼저 이전에 만든 두 캔버스 요소에서 사용자 지정 배경 이미지와 비디오 피드의 기본 픽셀 데이터를 검색합니다. 그런 다음 `ImageSegmenter`에서 제공한 마스크를 반복하여 전경에 있는 픽셀을 나타냅니다. 마스크를 반복하면서 사용자의 카메라 피드가 포함된 픽셀을 해당 배경 픽셀 데이터에 선택적으로 복사합니다. 이 작업이 완료되면 전경을 복사한 최종 픽셀 데이터를 배경으로 변환하고 캔버스에 그립니다.

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);
}
```

참고로 위의 모든 로직이 포함된 전체 `app.js` 파일은 다음과 같습니다.

JavaScript

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

// All helpers are expose on 'media-devices.js' and 'dom.js'
const { setupParticipant } = window;

const { Stage, LocalStageStream, SubscribeType, StageEvents, ConnectionState,
 StreamType } = IVSBroadcastClient;
const canvasElement = document.getElementById("canvas");
const background = document.getElementById("background");
const canvasCtx = canvasElement.getContext("2d");
const backgroundCtx = background.getContext("2d");
const video = document.getElementById("webcam");

import { ImageSegmenter, FilesetResolver } from "@mediapipe/tasks-vision";

let cameraButton = document.getElementById("camera-control");
let micButton = document.getElementById("mic-control");
let joinButton = document.getElementById("join-button");
let leaveButton = document.getElementById("leave-button");

let controls = document.getElementById("local-controls");
let audioDevicesList = document.getElementById("audio-devices");
let videoDevicesList = document.getElementById("video-devices");

// Stage management
let stage;
let joining = false;
let connected = false;
let localCamera;
let localMic;
let cameraStageStream;
let micStageStream;
let imageSegmenter;
let lastWebcamTime = -1;

const init = async () => {
  await initializeDeviceSelect();

  cameraButton.addEventListener("click", () => {
    const isMuted = !cameraStageStream.isMuted;
    cameraStageStream.setMuted(isMuted);
```

```
    cameraButton.innerText = isMuted ? "Show Camera" : "Hide Camera";
  });

  micButton.addEventListener("click", () => {
    const isMuted = !micStageStream.isMuted;
    micStageStream.setMuted(isMuted);
    micButton.innerText = isMuted ? "Unmute Mic" : "Mute Mic";
  });

  localCamera = await getCamera(videoDevicesList.value);
  const segmentationStream = canvasElement.captureStream();

  joinButton.addEventListener("click", () => {
    joinStage(segmentationStream);
  });

  leaveButton.addEventListener("click", () => {
    leaveStage();
  });

  initBackgroundCanvas();

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

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

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

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

  // Retrieve the User Media currently set on the page
  localMic = await getMic(audioDevicesList.value);

  cameraStageStream = new LocalStageStream(segmentationStream.getVideoTracks()[0]);
```

```
  micStageStream = new LocalStageStream(localMic.getAudioTracks()[0]);

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

  stage = new Stage(token, strategy);

  // Other available events:
  // https://aws.github.io/amazon-ivs-web-broadcast/docs/sdk-guides/stages#events
  stage.on(StageEvents.STAGE_CONNECTION_STATE_CHANGED, (state) => {
    connected = state === ConnectionState.CONNECTED;

    if (connected) {
      joining = false;
      controls.classList.remove("hidden");
    } else {
      controls.classList.add("hidden");
    }
  });

  stage.on(StageEvents.STAGE_PARTICIPANT_JOINED, (participant) => {
    console.log("Participant Joined:", participant);
  });

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

    let streamsToDisplay = streams;

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

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

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

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

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

  joining = false;
  connected = false;

  cameraButton.innerText = "Hide Camera";
  micButton.innerText = "Mute Mic";
  controls.classList.add("hidden");
};

function replaceBackground(result) {
  let imageData = canvasCtx.getImageData(0, 0, video.videoWidth,
 video.videoHeight).data;
  let backgroundData = backgroundCtx.getImageData(0, 0, video.videoWidth,
 video.videoHeight).data;
  const mask = result.categoryMask.getAsFloat32Array();
  let j = 0;

  for (let i = 0; i < mask.length; ++i) {
    const maskVal = Math.round(mask[i] * 255.0);

    j += 4;
    if (maskVal < 255) {
      backgroundData[j] = imageData[j];
```

```
      backgroundData[j + 1] = imageData[j + 1];
      backgroundData[j + 2] = imageData[j + 2];
      backgroundData[j + 3] = imageData[j + 3];
    }
  }
  const uint8Array = new Uint8ClampedArray(backgroundData.buffer);
  const dataNew = new ImageData(uint8Array, video.videoWidth, video.videoHeight);
  canvasCtx.putImageData(dataNew, 0, 0);
  window.requestAnimationFrame(renderVideoToCanvas);
}

const createImageSegmenter = async () => {
  const audio = await FilesetResolver.forVisionTasks("https://cdn.jsdelivr.net/npm/
@mediapipe/tasks-vision@0.10.2/wasm");

  imageSegmenter = await ImageSegmenter.createFromOptions(audio, {
    baseOptions: {
      modelAssetPath: "https://storage.googleapis.com/mediapipe-models/image_segmenter/
selfie_segmenter/float16/latest/selfie_segmenter.tflite",
      delegate: "GPU",
    },
    runningMode: "VIDEO",
    outputCategoryMask: true,
  });
};

const renderVideoToCanvas = async () => {
  if (video.currentTime === lastWebcamTime) {
    window.requestAnimationFrame(renderVideoToCanvas);
    return;
  }
  lastWebcamTime = video.currentTime;
  canvasCtx.drawImage(video, 0, 0, video.videoWidth, video.videoHeight);

  if (imageSegmenter === undefined) {
    return;
  }

  let startTimeMs = performance.now();

  imageSegmenter.segmentForVideo(video, startTimeMs, replaceBackground);
};

const initBackgroundCanvas = () => {
```

```
  let img = new Image();
  img.src = "beach.jpg";

  img.onload = () => {
    backgroundCtx.clearRect(0, 0, canvas.width, canvas.height);
    backgroundCtx.drawImage(img, 0, 0);
  };
};

createImageSegmenter();
init();
```

Webpack 구성 파일 생성

이 구성을 Webpack 구성 파일에 추가하여 `app.js`을 번들링하면 가져오기 호출이 제대로 작동합니다.

JavaScript

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

JavaScript 파일 번들링

```
npm run build
```

`index.html`이 들어 있는 디렉토리에서 간단한 HTTP 서버를 시작하고 `localhost:8000`을 열어서 결과를 확인합니다.

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

Android

실시간 스트림의 배경을 바꾸기 위해 Google ML Kit의 셀카 분할 API를 사용할 수 있습니다. 셀카 분할 API는 카메라 이미지를 입력으로 받아들이고 이미지의 각 픽셀에 대한 신뢰도 점수를 제공하는 마스

크를 반환합니다. 이를 통해 이미지가 전경에 있었는지 배경에 있었는지 알 수 있습니다. 그런 다음 신뢰도 점수를 기반으로 배경 이미지 또는 전경 이미지에서 해당 픽셀 색상을 검색할 수 있습니다. 이 프로세스는 마스크의 모든 신뢰도 점수를 검사할 때까지 계속됩니다. 그 결과 전경 픽셀과 배경 이미지의 픽셀을 포함한 새로운 픽셀 색상 배열이 만들어집니다.

배경 교체를 IVS 실시간 스트리밍 Android 브로드캐스트 SDK와 통합하려면 다음 작업을 수행해야 합니다.

1. CameraX 라이브러리와 Google ML Kit를 설치합니다.
2. 표준 문안 변수를 초기화합니다.
3. 사용자 지정 이미지 소스를 생성합니다.
4. 카메라 프레임을 관리합니다.
5. 카메라 프레임을 Google ML Kit로 전달하세요.
6. 카메라 프레임 전경을 사용자 지정 배경에 오버레이하세요.
7. 새 이미지를 사용자 지정 이미지 소스에 제공하세요.

CameraX 라이브러리 및 Google ML Kit 설치

라이브 카메라 피드에서 이미지를 추출하려면 Android의 CameraX 라이브러리를 사용하세요. CameraX 라이브러리와 Google ML Kit를 설치하려면 모듈의 `build.gradle` 파일에 다음을 추가하세요. `${camerax_version}`과 `${google_ml_kit_version}`을 각각 최신 버전의 CameraX 및 Google ML Kit 라이브러리로 교체하세요.

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}"
```

다음 라이브러리를 가져옵니다.

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
```

표준 문안 변수 초기화

ImageAnalysis의 인스턴스와 다음 ExecutorService의 인스턴스를 초기화합니다.

Java

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

STREAM_MODE에서 Segmenter 인스턴스를 초기화하세요.

Java

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

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

사용자 지정 이미지 소스 생성

활동의 onCreate 메서드에서 DeviceDiscovery 개체의 인스턴스를 생성하고 사용자 지정 이미지 소스를 생성하세요. 사용자 지정 이미지 소스에서 제공한 Surface로 사용자 지정 배경 이미지 위에 전경이 오버레이된 최종 이미지를 받게 됩니다. 그런 다음 사용자 지정 이미지 소스를 사용하여 ImageLocalStageStream의 인스턴스를 생성합니다. 그러면 ImageLocalStageStream의 인스턴스(이 예제에서는 filterStream으로 이름이 지정됨)를 스테이지에 게시할 수 있습니다. 스테이지 설정에 대한 지침은 IVS Android 브로드캐스트 SDK 가이드를 참조하세요. 마지막으로 카메라를 관리하는 데 사용할 스레드도 생성하세요.

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()
```

카메라 프레임 관리

다음으로 카메라를 초기화하는 함수를 생성합니다. 이 함수는 CameraX 라이브러리를 사용하여 라이브 카메라 피드에서 이미지를 추출합니다. 먼저 `cameraProviderFuture`라는 `ProcessCameraProvider` 인스턴스를 생성합니다. 이 객체는 카메라 공급자를 확보한 미래의 결과를 나타냅니다. 그런 다음 프로젝트에서 이미지를 비트맵으로 로드합니다. 이 예제에서는 해변 이미지를 배경으로 사용하지만 원하는 어떤 이미지라도 사용할 수 있습니다.

그런 다음 `cameraProviderFuture`에 리스너를 추가합니다. 카메라를 사용할 수 있게 되거나 카메라 공급자를 구하는 과정에서 오류가 발생하면 이 리스너에 알림이 전송됩니다.

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))
    }
```

리스너 내에서 라이브 카메라 피드의 각 개별 프레임에 액세스할 수 있도록 ImageAnalysis.Builder를 생성하세요. 백 프레셔 전략을 STRATEGY_KEEP_ONLY_LATEST로 설정합니다. 이렇게 하면 한 번에 하나의 카메라 프레임만 처리에 전송되도록 할 수 있습니다. 각 개별 카메라 프레임을 비트맵으로 변환하면 픽셀을 추출하여 나중에 사용자 지정 배경 이미지와 결합할 수 있습니다.

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())
```

카메라 프레임을 Google ML Kit로 전달

다음으로, InputImage를 생성하여 처리를 위한 Segmenter 인스턴스에 전달합니다. ImageAnalysis의 인스턴스에서 제공하는 ImageProxy로 InputImage를 생성할 수 있습니다. Segmenter에 InputImage가 제공되면 픽셀이 전경이나 배경에 있을 가능성을 나타내는 신뢰도 점수가 포함된 마스크를 반환합니다. 이 마스크는 너비 및 높이 속성도 제공하며, 이 속성을 사용하여 이전에 로드한 사용자 지정 배경 이미지의 배경 픽셀을 포함하는 새 배열을 생성할 수 있습니다.

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)
```

카메라 프레임 전경을 사용자 지정 배경에 오버레이

신뢰도 점수가 포함된 마스크, 비트맵의 카메라 프레임, 사용자 지정 배경 이미지의 색상 픽셀을 사용하면 전경을 사용자 지정 배경에 오버레이하는 데 필요한 모든 것을 얻을 수 있습니다. 그 다음 파라미터로 overlayForeground 함수를 호출합니다.

Java

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

이 함수는 마스크를 반복하고 신뢰도 값을 확인하여 배경 이미지에서 해당 픽셀 색상을 가져올지 아니면 카메라 프레임에서 가져올지 결정합니다. 신뢰도 값이 마스크의 픽셀이 배경에 있을 가능성이 높다는 것을 나타내면 배경 이미지에서 해당 픽셀 색상을 가져오고, 그렇지 않으면 카메라 프레임에서 해당 픽셀 색상을 가져와 전경을 만듭니다. 함수가 마스크 반복을 마치면 새 색상 픽셀 배열을 사용하여 새 비트맵을 생성하고 반환합니다. 이 새 비트맵은 사용자 지정 배경에 오버레이된 전경을 포함하고 있습니다.

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
        )
    }
```

새 이미지를 사용자 지정 이미지 소스에 제공

그런 다음 사용자 지정 이미지 소스에서 제공한 Surface에 새 비트맵을 쓸 수 있습니다. 그러면 스테이지로 브로드캐스트합니다.

Java

```
resultBitmap = overlayForeground(mask, inputBitmap, mutableBitmap, bgBitmap)
```

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

카메라 프레임을 가져와서 Segmenter로 전달하고 백그라운드에 오버레이하는 전체 기능은 다음과 같습니다.

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 브로드캐스트 SDK: 모바일 오디오 모드(실시간 스트리밍)

오디오 품질은 실제 팀 미디어 경험의 중요한 부분이며, 모든 사용 사례에 가장 적합한 단일 오디오 구성은 없습니다. 사용자가 IVS 실시간 스트림을 청취할 때 최상의 경험을 할 수 있도록 모바일 SDK는 몇 가지 사전 설정 오디오 구성과 필요에 따라 더욱 강력한 사용자 지정을 제공합니다.

소개

IVS 모바일 브로드캐스트 SDK는 StageAudioManager 클래스를 제공합니다. 이 클래스는 두 플랫폼에서 기본 오디오 모드를 제어할 수 있는 단일 접점이 되도록 설계되었습니다. Android에서 이를 통해 오디오 모드, 오디오 소스, 콘텐츠 유형, 사용량, 통신 디바이스 등 AudioManager를 제어합니다. iOS에서 애플리케이션 AVAudioSession을 제어하고 voiceProcessing의 활성화 여부도 제어합니다.

중요: IVS 실시간 브로드캐스트 SDK가 활성화되어 있는 동안에는 AudioManager 또는 AVAudioSession와 직접 상호 작용하지 마세요. 이렇게 하면 오디오가 손실되거나 오디오가 잘못된 디바이스에서 녹음되거나 재생될 수 있습니다.

첫 번째 DeviceDiscovery 또는 Stage 객체를 생성하기 전에 StageAudioManager 클래스를 구성해야 합니다.

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
```

DeviceDiscovery 또는 Stage 인스턴스를 초기화하기 전에 StageAudioManager에 아무것도 설정하지 않은 경우 VideoChat 사전 설정이 자동으로 적용됩니다.

오디오 모드 사전 설정

실시간 브로드캐스트 SDK는 아래 설명과 같이 일반적인 사용 사례에 맞게 조정된 세 가지 사전 설정을 제공합니다. 각 사전 설정에 대해 사전 설정을 서로 차별화하는 다섯 가지 주요 카테고리를 다룹니다.

비디오 채팅

로컬 디바이스가 다른 참가자와 실시간으로 대화할 때 사용하도록 설계된 기본 사전 설정입니다.

범주	Android	iOS
에코 소거	활성화됨	활성화됨
볼륨 로커	호출 볼륨	호출 볼륨
마이크 선택	OS에 따라 제한됩니다. USB 마이크를 사용할 수 없을 수도 있습니다.	OS에 따라 제한됩니다. USB 및 Bluetooth 마이크를 사용할 수 없을 수도 있습니다. 입력과 출력을 함께 처리하는 Bluetooth 헤드셋이 작동해야 합니다(예: AirPods).
오디오 출력	모든 출력 디바이스가 작동해야 합니다.	OS에 따라 제한됩니다. 유선 헤드셋은 사용할 수 없을 수도 있습니다.
오디오 품질	중간/낮음 미디어 재생이 아니라 전화 통화처럼 들릴 것입니다.	중간/낮음 미디어 재생이 아니라 전화 통화처럼 들릴 것입니다.

구독 전용

이 사전 설정은 다른 게시 참가자를 구독하고 직접 게시하지는 않으려는 경우에 적합합니다. 오디오 품질에 초점을 맞추고 사용 가능한 모든 출력 디바이스를 지원합니다.

범주	Android	iOS
에코 소거	Disabled(비활성)	Disabled(비활성)

범주	Android	iOS
볼륨 로커	미디어 볼륨	미디어 볼륨
마이크 선택	해당 없음, 이 사전 설정은 게시용으로 설계되지 않았습니다.	해당 없음, 이 사전 설정은 게시용으로 설계되지 않았습니다.
오디오 출력	모든 출력 디바이스가 작동해야 합니다.	모든 출력 디바이스가 작동해야 합니다.
오디오 품질	높음 음악을 포함한 모든 미디어 유형은 선명하게 전달되어야 합니다.	높음 음악을 포함한 모든 미디어 유형은 선명하게 전달되어야 합니다.

Studio

이 사전 설정은 게시 기능은 그대로 유지하면서 고품질 구독을 할 수 있도록 설계되었습니다. 에코 소거 기능을 제공하려면 레코딩 및 재생 하드웨어가 필요합니다. 여기 사용 사례에서는 USB 마이크와 유선 헤드셋을 사용합니다. SDK는 이러한 디바이스가 에코를 발생시키지 않도록 물리적으로 분리함으로써 최고 품질의 오디오를 유지합니다.

범주	Android	iOS
에코 소거	Disabled(비활성)	Disabled(비활성)
볼륨 로커	대부분의 경우에는 미디어 볼륨입니다. Bluetooth 마이크 연결 시 통화 볼륨입니다.	미디어 볼륨
마이크 선택	모든 마이크가 작동해야 합니다.	모든 마이크가 작동해야 합니다.
오디오 출력	모든 출력 디바이스가 작동해야 합니다.	모든 출력 디바이스가 작동해야 합니다.
오디오 품질	높음 양쪽 모두 음악을 전송하고 반대편에서도 선명하게 들을 수 있어야 합니다.	높음 양쪽 모두 음악을 전송하고 반대편에서도 선명하게 들을 수 있어야 합니다. Bluetooth 헤드셋을 연결하면 헤드셋에 따라 Bluetooth SCO 모드가

범주	Android	iOS
	Bluetooth 헤드셋을 연결하면 Bluetooth SCO 모드가 활성화되면 오디오 품질이 떨어집니다.	활성화되어 오디오 음질이 떨어질 수 있습니다.

고급 사용 사례

사전 설정 외에도 iOS 및 Android 실시간 스트리밍 브로드캐스트 SDK를 통해 기본 플랫폼 오디오 모드를 구성할 수 있습니다.

- Android에서 AudioSource, Usage 및 ContentType을 설정합니다.
- iOS에서 AVAudioSession.Category, AVAudioSession.CategoryOptions, AVAudioSession.Mode 그리고 게시하는 동안 음성 처리가 활성화되었는지 여부를 전환하는 기능을 사용할 수 있습니다.

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
```

Android에서 Bluetooth로 게시

다음 조건이 충족되면 SDK가 Android의 `VIDEO_CHAT` 사전 설정을 자동으로 되돌립니다.

- 할당된 구성은 `VOICE_COMMUNICATION` 사용량 값을 사용하지 않습니다.
- Bluetooth 마이크가 디바이스에 연결되어 있습니다.
- 로컬 참가자가 스테이지에 게시하고 있습니다.

이는 오디오 레코딩에 Bluetooth 헤드셋을 사용하는 방식과 관련된 Android 운영 체제의 제한 사항입니다.

다른 SDK와 통합

iOS와 Android 모두 애플리케이션당 하나의 활성 오디오 모드만 지원하기 때문에 애플리케이션에서 오디오 모드를 제어해야 하는 여러 SDK를 사용하는 경우 종종 충돌이 발생합니다. 이러한 충돌이 발생하는 경우 아래에서 설명한 몇 가지 일반적인 해결 전략을 시도해 볼 수 있습니다.

오디오 모드 값 일치

IVS SDK의 고급 오디오 구성 옵션이나 다른 SDK의 기능을 사용하여 두 SDK를 기본 값에 맞춰 정렬합니다.

Agora

iOS

iOS에서 Agora SDK에 `AVAudioSession` 활성 상태를 유지하도록 지시하면 IVS 실시간 스트리밍 브로드캐스트 SDK가 사용하는 동안 Agora SDK가 비활성화되지 않습니다.

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

Android

`RtcEngine`에 `setEnableSpeakerphone`을 호출하지 않고 IVS 실시간 스트리밍 브로드캐스트 SDK로 게시하는 동안 `enableLocalAudio(false)`을 호출해 보세요. IVS SDK가 게시되지 않을 때 다시 `enableLocalAudio(true)`를 호출할 수 있습니다.

IVS 실시간 스트리밍과 함께 Amazon EventBridge 사용

Amazon EventBridge를 사용하여 Amazon Interactive Video Service(IVS) 스트림을 모니터링할 수 있습니다.

Amazon IVS는 스트림의 상태에 대한 변경 이벤트를 Amazon EventBridge로 전송합니다. 전송되는 모든 이벤트가 유효합니다. 다만 이벤트는 '최선형 전송'을 기반으로 합니다. 즉, 다음을 보장하지 않습니다.

- 이벤트 전송 - 지정된 이벤트(예: 참가자가 게시함)가 발생하더라도 Amazon IVS가 해당 이벤트를 EventBridge로 보내지 않을 수도 있습니다. 이벤트가 전송이 될 때까지 Amazon IVS는 수 시간동안 이벤트 전송을 시도합니다.
- 전송되는 이벤트는 지정된 기간에 도착합니다. 최대 몇 시간 이전의 이벤트도 수신할 수 있습니다.
- 순서대로 이벤트 전송 - 특히, 이벤트가 서로 짧은 간격 안에 전송되는 경우 이벤트 순서가 잘못될 수 있습니다. 예를 들어, 참가자가 게시함 전에 참가자가 게시 취소함을 볼 수 있습니다.

이벤트가 누락되거나 늦거나 순서가 맞지 않는 경우는 드물지만 알림 이벤트의 순서나 존재 여부에 종속되는 비즈니스에 중요한 프로그램을 작성하는 경우 이러한 가능성을 염두에 두고 처리해야 합니다.

다음 이벤트 중 하나에 대한 Eventbridge 규칙을 생성할 수 있습니다.

이벤트 유형	이벤트	전송된 시간
IVS 구성 상태 변경	대상 실패	대상에 출력하려는 시도가 실패했습니다. 예를 들어 스트림 키가 없거나 다른 브로드캐스트가 진행 중이어서 채널에 브로드캐스팅이 실패했습니다.
IVS 구성 상태 변경	대상 시작	대상에 출력이 성공적으로 시작되었습니다.
IVS 구성 상태 변경	대상 종료	대상에 출력이 완료되었습니다.
IVS 구성 상태 변경	대상 재연결	대상에 출력이 중단되어 재연결을 시도하고 있습니다.

이벤트 유형	이벤트	전송된 시간
IVS 구성 상태 변경	세션 시작	구성 세션을 생성했습니다. 이 이벤트는 구성 프로세스 파이프라인이 성공적으로 초기화될 때 발생합니다. 현재 구성 파이프라인은 스테이지를 성공적으로 구독했으며 미디어를 수신하고 비디오를 작성할 수 있습니다.
IVS 구성 상태 변경	세션 종료	구성 세션이 완료되었습니다.
IVS 구성 상태 변경	세션 실패	스테이지 리소스를 사용할 수 없거나 기타 내부 오류로 인해 구성 파이프라인이 초기화에 실패했습니다.
IVS 스테이지 업데이트	참가자가 게시함	참가자가 스테이지에 게시를 시작합니다.
IVS 스테이지 업데이트	참가자가 게시 취소함	참가자가 스테이지에 게시를 중지했습니다.

Amazon IVS에 대한 Amazon EventBridge 규칙 생성

Amazon IVS에서 생성되는 이벤트에서 트리거되는 규칙을 생성할 수 있습니다. Amazon EventBridge 사용 설명서의Create a rule in Amazon EventBridge 단계를 따르세요. 서비스를 선택할 때 Interactive Video Service(IVS)를 선택합니다.

예제: 구성 상태 변경

대상 실패: 이 이벤트는 대상에 출력하려는 시도가 실패했을 때 전송됩니다. 예를 들어 스트림 키가 없거나 다른 브로드캐스트가 진행 중이어서 채널에 브로드캐스팅이 실패했습니다.

```
{
   "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"
   }
}
```

대상 시작: 이 이벤트는 대상에 출력이 성공적으로 시작되었을 때 전송됩니다.

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

대상 종료: 이 이벤트는 대상에 출력이 완료되면 전송됩니다.

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

대상 재연결: 이 이벤트는 대상에 출력이 중단되어 재연결을 시도할 때 전송됩니다.

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

세션 시작: 이 이벤트는 구성 세션이 생성될 때 전송됩니다. 이 이벤트는 구성 프로세스 파이프라인이 성공적으로 초기화될 때 발생합니다. 현재 구성 파이프라인은 스테이지를 성공적으로 구독했으며 미디어를 수신하고 비디오를 작성할 수 있습니다.

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

세션 종료: 이 이벤트는 구성 세션이 완료되고 모든 리소스가 삭제되었을 때 전송됩니다.

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

세션 실패: 이 이벤트는 스테이지 리소스를 사용할 수 없거나, 스테이지에 참가자가 없거나, 기타 내부 오류로 인해 구성 파이프라인 초기화에 실패했을 때 전송됩니다.

```
{
   "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"
   }
}
```

예: 스테이지 업데이트

스테이지 업데이트 이벤트에는 이벤트를 분류하는 이벤트 이름과 해당 이벤트에 대한 메타데이터가 포함됩니다. 메타데이터에는 이벤트를 트리거한 참가자 ID, 연결된 스테이지 및 세션 ID, 사용자 ID가 포함됩니다.

참가자가 게시함: 이 이벤트는 참가자가 스테이지에 게시를 시작할 때 전송됩니다.

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

참가자가 게시 취소함: 이 이벤트는 참가자가 스테이지에 게시를 중지했을 때 전송됩니다.

```
{
    "version": "0",
    "id": "12345678-1a23-4567-a1bc-1a2b34567890",
    "detail-type": "IVS Stage Update",
    "source": "aws.ivs",
    "account": "123456789012",
```

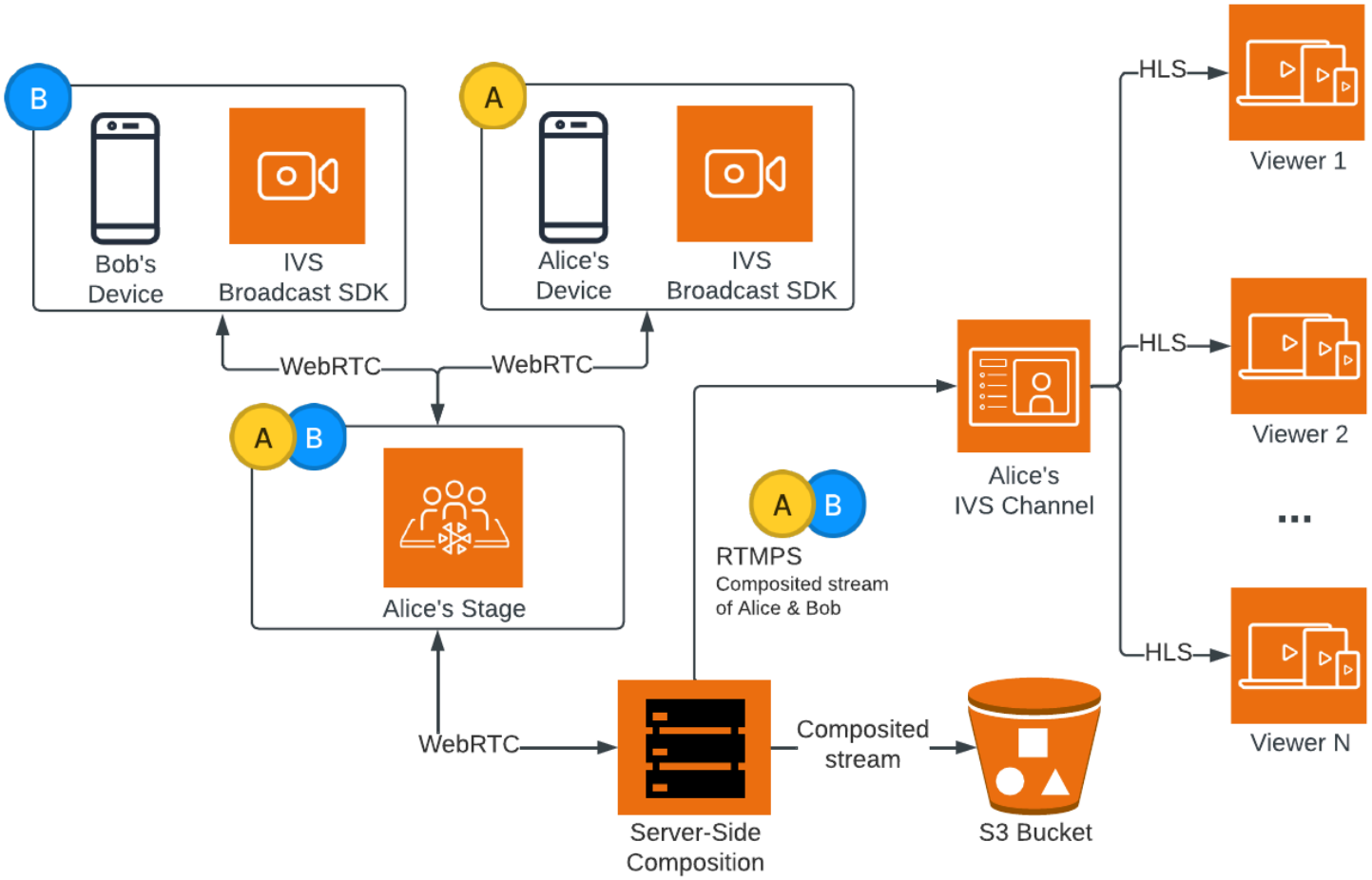
```
    "time": "2020-06-23T20:12:36Z",
    "region": "us-west-2",
    "resources": [
        "arn:aws:ivs:us-west-2:123456789012:stage/AbCdef1G2hij"
    ],
    "detail": {
        "session_id": "st-1234567890",
        "event_name": "Participant Unpublished",
        "user_id": "Your User Id",
        "participant_id": "xYz1c2d3e4f"
    }
}
```

서버 측 구성(실시간 스트리밍)

서버 측 구성은 IVS 서버를 사용하여 모든 스테이지 참가자의 오디오와 비디오를 믹싱한 다음 이 믹스된 비디오를 IVS 채널(예: 더 많은 시청자에게 도달하기 위해) 또는 S3 버킷으로 보냅니다. 서버 측 구성은 스테이지의 홈 리전에서 IVS 컨트롤 플레인 엔드포인트를 통해 호출됩니다.

서버 측 구성을 사용하여 스테이지를 브로드캐스팅하거나 레코딩하면 많은 이점이 있으므로 효율적이고 안정적인 클라우드 기반 비디오 워크플로를 찾는 사용자에게 매력적인 선택입니다.

다음 다이어그램은 서버 측 구성의 작동 방식을 보여줍니다.



이점

클라이언트 측 구성과 비교할 때 서버 측 구성은 다음과 같은 이점이 있습니다.

- 클라이언트 부하 감소 - 서버 측 구성을 사용하면 오디오 및 비디오 소스를 처리하고 결합하는 부담이 개별 클라이언트 디바이스에서 서버 자체로 이동합니다. 서버 측 구성을 사용하면 클라이언트 디

바이스에서 보기를 합성하고 IVS로 전송하는 데 클라이언트 디바이스의 CPU와 네트워크 리소스를 사용할 필요가 없습니다. 즉, 시청자는 디바이스에서 리소스를 많이 사용하는 작업을 처리하지 않고도 브로드캐스트를 시청할 수 있으므로 배터리 수명이 향상되고 시청 환경이 개선됩니다.

- 일관된 품질 - 서버 측 구성을 통해 최종 스트림의 품질, 해상도 및 비트레이트를 정밀하게 제어할 수 있습니다. 이를 통해 개별 디바이스의 성능과 상관없이 모든 시청자에게 일관된 시청 경험을 보장합니다.
- 복원력 - 구성 프로세스를 서버에서 중앙 집중화함으로써 브로드캐스트가 더욱 강력해집니다. 게시자 디바이스에 기술적 제한 사항이나 변동이 있더라도 서버는 적응하여 모든 대상에게 더 매끄러운 스트림을 제공할 수 있습니다.
- 대역폭 효율성 - 서버가 구성을 처리하므로 스테이지 게시자는 비디오를 IVS로 브로드캐스트하는 데 추가 대역폭을 소비하지 않아도 됩니다.

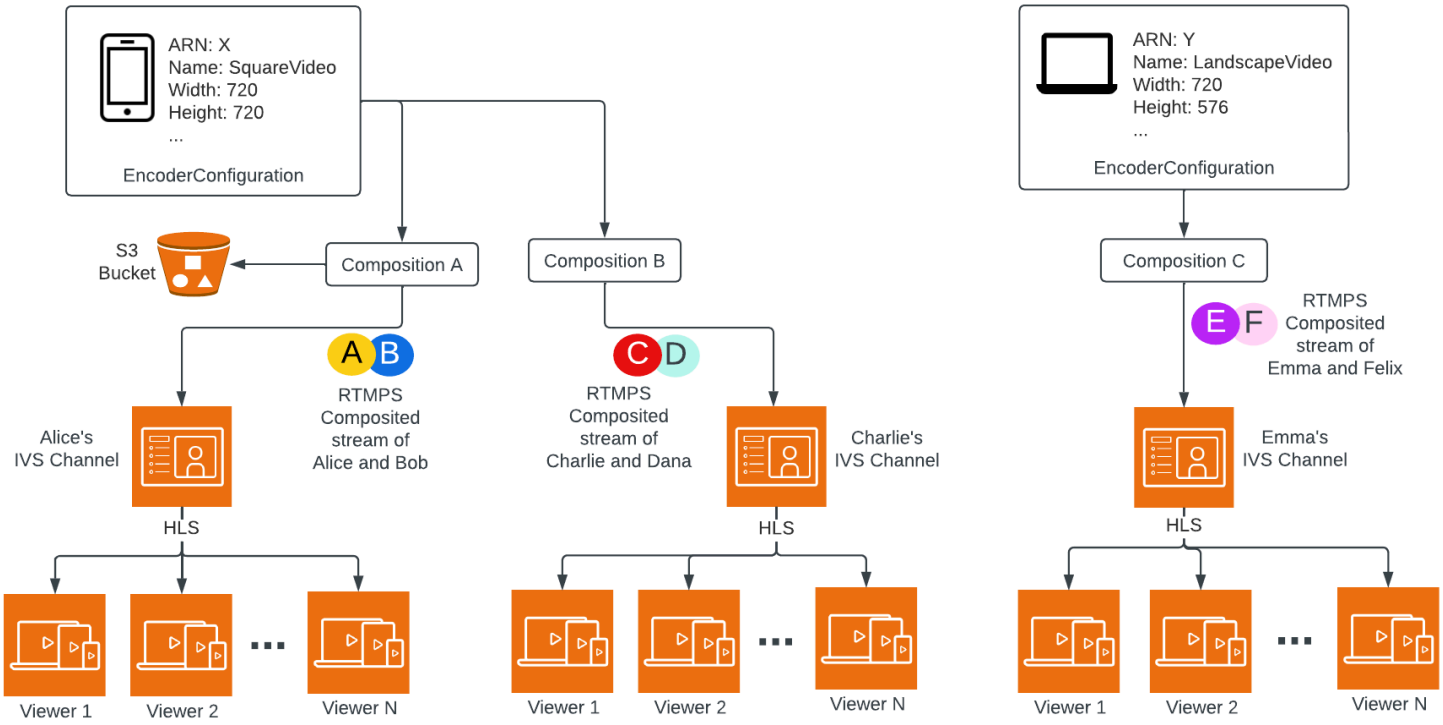
그 대신 IVS 채널로 스테이지를 브로드캐스트하려면 구성 클라이언트 측에서 할 수 있습니다. IVS 저지연 스트리밍 사용 설명서의 IVS 스트림에서 다중 호스트 활성화를 참조하세요.

IVS API

서버 측 구성은 다음과 같은 주요 API 요소를 사용합니다.

- EncoderConfiguration 객체를 사용하면 생성할 비디오의 형식(높이, 너비, 비트레이트 및 기타 스트리밍 파라미터)을 사용자 지정할 수 있습니다. StartComposition 엔드포인트를 호출할 때마다 EncoderConfiguration을 재사용할 수 있습니다.
- 구성 엔드포인트는 비디오 구성을 추적하고 IVS 채널로 출력합니다.
- StorageConfiguration은 구성을 레코드하는 S3 버킷을 추적합니다.

서버 측 구성을 사용하려면 StartComposition 엔드포인트를 호출할 때 EncoderConfiguration을 생성하고 이를 연결해야 합니다. 이 예제에서는 SquareVideo EncoderConfiguration이 두 개의 구성에 사용됩니다.

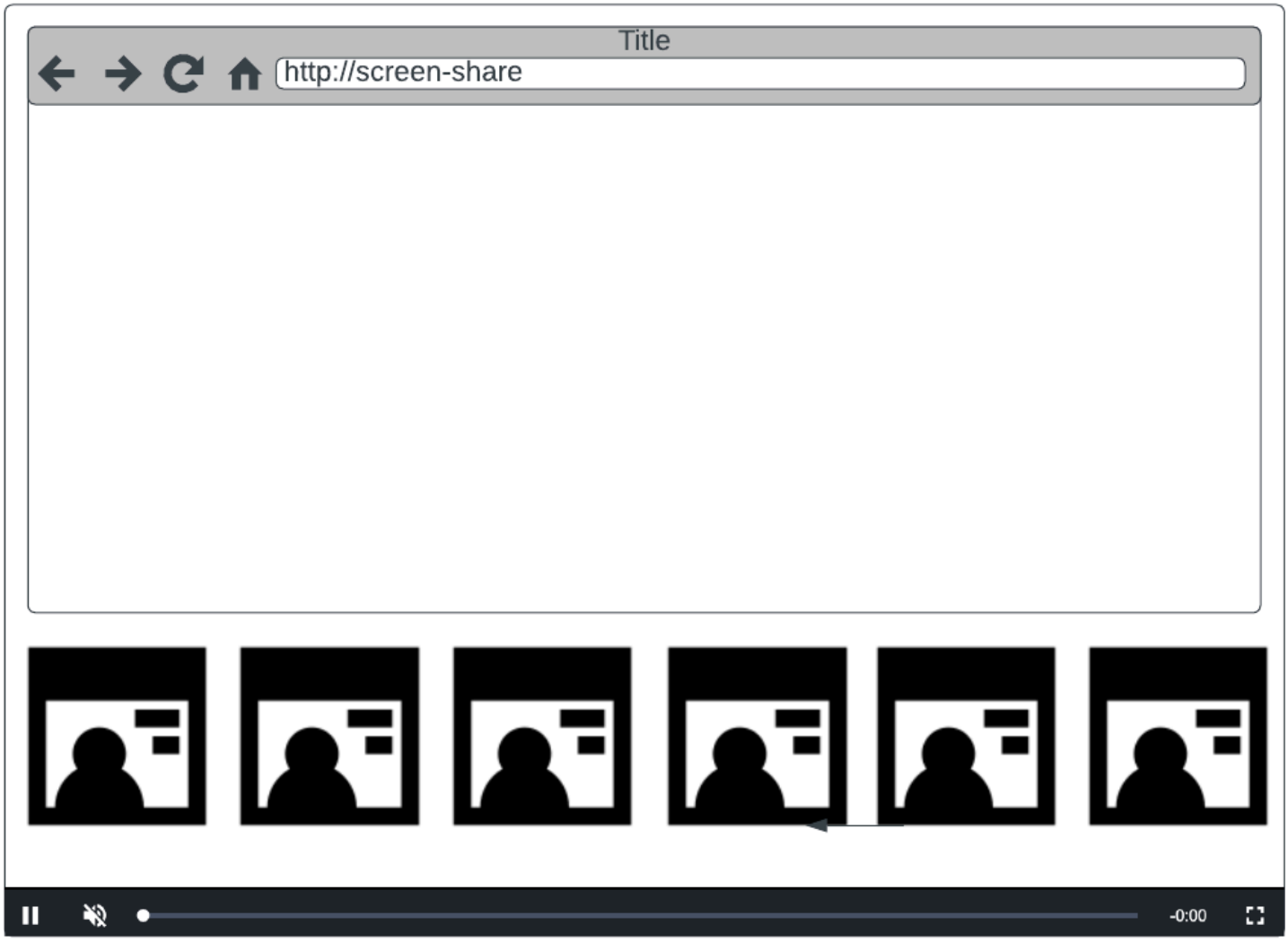


전체 내용은 [IVS 실시간 스트리밍 API 참조]를 참조하세요.

레이아웃

기본적으로 서버 측 구성 기능은 그리드 레이아웃을 사용하여 스테이지 참가자를 동일한 크기의 슬롯에 정렬합니다.

이 레이아웃은 고객이 추천 슬롯을 구성하고 호출할 수 있는 옵션을 제공합니다. 추천 슬롯은 메인 화면에 있으며, 다른 참가자들은 같은 크기의 슬롯으로 그 아래에 표시됩니다.



참고: 서버 측 구성에서 스테이지 게시자가 지원하는 최대 해상도는 1080p입니다. 게시자가 1080p 이상의 비디오를 전송하는 경우 게시자는 오디오 전용 참가자로 렌더링됩니다.

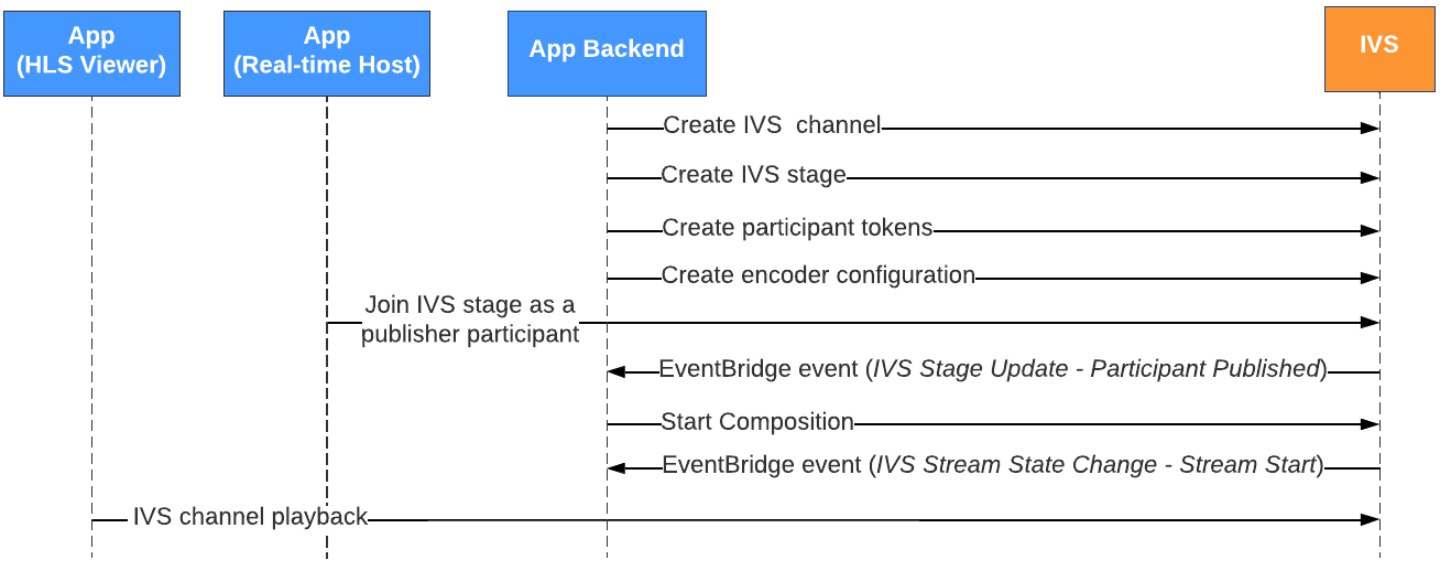
시작하기

필수 조건

서버 측 구성을 사용하려면 활성 게시자가 있는 스테이지가 있어야 하며 IVS 채널 및/또는 S3 버킷을 구성 대상으로 사용해야 합니다. 아래에서는 참가자가 게시할 때 EventBridge 이벤트를 사용하여 IVS 채널에 스테이지를 브로드캐스트하는 구성을 시작하는 한 가지 가능한 워크플로를 설명합니다. 그 대신 자체 앱 로직에 따라 구성을 시작하고 중지할 수 있습니다. 서버 측 구성을 사용하여 스테이지를 S3 버킷에 직접 레코드하는 방법을 보여주는 또 다른 예제는 복합 레코딩을 참조하세요.

1. IVS 채널을 생성하세요. Amazon IVS 저지연 스트리밍 시작하기를 참조하세요.

2. 각 게시자를 위한 IVS 스테이지와 참가자 토큰을 생성하세요.
3. EncoderConfiguration을 생성하세요.
4. 스테이지에 참여하고 스테이지에 게시하세요. (실시간 스트리밍 브로드캐스트 SDK 가이드의 "게시 및 구독" 섹션: 웹, Android, iOS를 참조하세요.)
5. 참가자가 게시한 EventBridge 이벤트를 받으면 StartComposition을 호출하세요.
6. 몇 초간 기다린 후 채널 재생에서 합성된 보기를 확인하세요.



참고: 스테이지에 있는 게시자 참가자가 60초 동안 활동이 없으면 구성이 자동 종료됩니다. 이때 구성이 종료되고 STOPPED 상태로 전환합니다. STOPPED 상태로 몇 분 지나면 구성을 자동으로 삭제합니다.

CLI 지침

AWS CLI를 사용하는 것은 고급 옵션이며, 먼저 시스템에 CLI를 다운로드하고 구성해야 합니다. 자세한 내용은 AWS 명령줄 인터페이스 사용 설명서를 참조하세요.

이제 CLI를 사용하여 리소스를 생성하고 관리할 수 있습니다. 구성 엔드포인트는 `ivs-realtime` 네임스페이스 아래에 있습니다.

EncoderConfiguration 리소스 생성

EncoderConfiguration은 생성된 비디오의 형식(높이, 너비, 비트레이트 및 기타 스트리밍 파라미터)을 사용자 지정할 수 있는 객체입니다. 다음 단계에서 설명하는 대로 Composition 엔드포인트를 호출할 때마다 EncoderConfiguration을 재사용할 수 있습니다.

아래 명령은 비디오 비트레이트, 프레임 속도 및 해상도와 같은 서버 측 비디오 구성 파라미터를 구성하는 EncoderConfiguration 리소스를 생성합니다.

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

다음과 같이 응답합니다.

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

구성 시작

위 응답에 제공된 EncoderConfiguration ARN을 사용하여 구성 리소스를 생성합니다.

```
aws ivs-realtime start-composition --stage-arn "arn:aws:ivs:us-
east-1:927810967299:stage/8faHz1SQp0ik" --destinations  '[{"channel": {"channelArn":
 "arn:aws:ivs:us-east-1:927810967299:channel/DOlMW4dfMR8r", "encoderConfigurationArn":
 "arn:aws:ivs:us-east-1:927810967299:encoder-configuration/9W59OBY2M8s4"}}]'
```

응답은 구성이 `STARTING` 상태로 생성되었음을 보여줍니다. 구성이 구성을 게시하기 시작하면 상태가 `ACTIVE`로 전환됩니다. (ListCompositions 또는 GetComposition 엔드포인트를 호출하여 상태를 확인할 수 있습니다.)

구성이 `ACTIVE`가 되면 IVS 채널에서 ListCompositions를 사용하여 IVS 스테이지의 복합 보기를 볼 수 있습니다.

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

다음과 같이 응답합니다.

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

참고: 구성을 계속 유지하려면 게시자 참가자가 스테이지에 적극적으로 게시하도록 해야 합니다. 자세한 내용은 실시간 스트리밍 브로드캐스트 SDK 가이드의 "게시 및 구독" 섹션(웹, Android, iOS)을 참조하세요. 각 참가자에 대해 별도의 스테이지 토큰을 생성해야 합니다.

화면 공유 사용

고정 화면 공유 레이아웃을 사용하려면 아래 단계를 따르세요.

EncoderConfiguration 리소스 생성

아래 명령은 서버 측 구성 파라미터(비디오 비트레이트, 프레임 속도 및 해상도)를 구성하는 EncoderConfiguration 리소스를 생성합니다.

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

`screen-share` 속성이 있는 스테이지 참가자 토큰을 생성합니다. `featured` 슬롯 이름으로 `screen-share`를 지정할 것이므로 `true` 속성이 `screen-share`로 설정된 스테이지 토큰을 생성해야 합니다.

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

다음과 같이 응답합니다.

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

구성 시작

화면 공유 기능을 사용하여 구성을 시작하려면 다음 명령을 사용합니다.

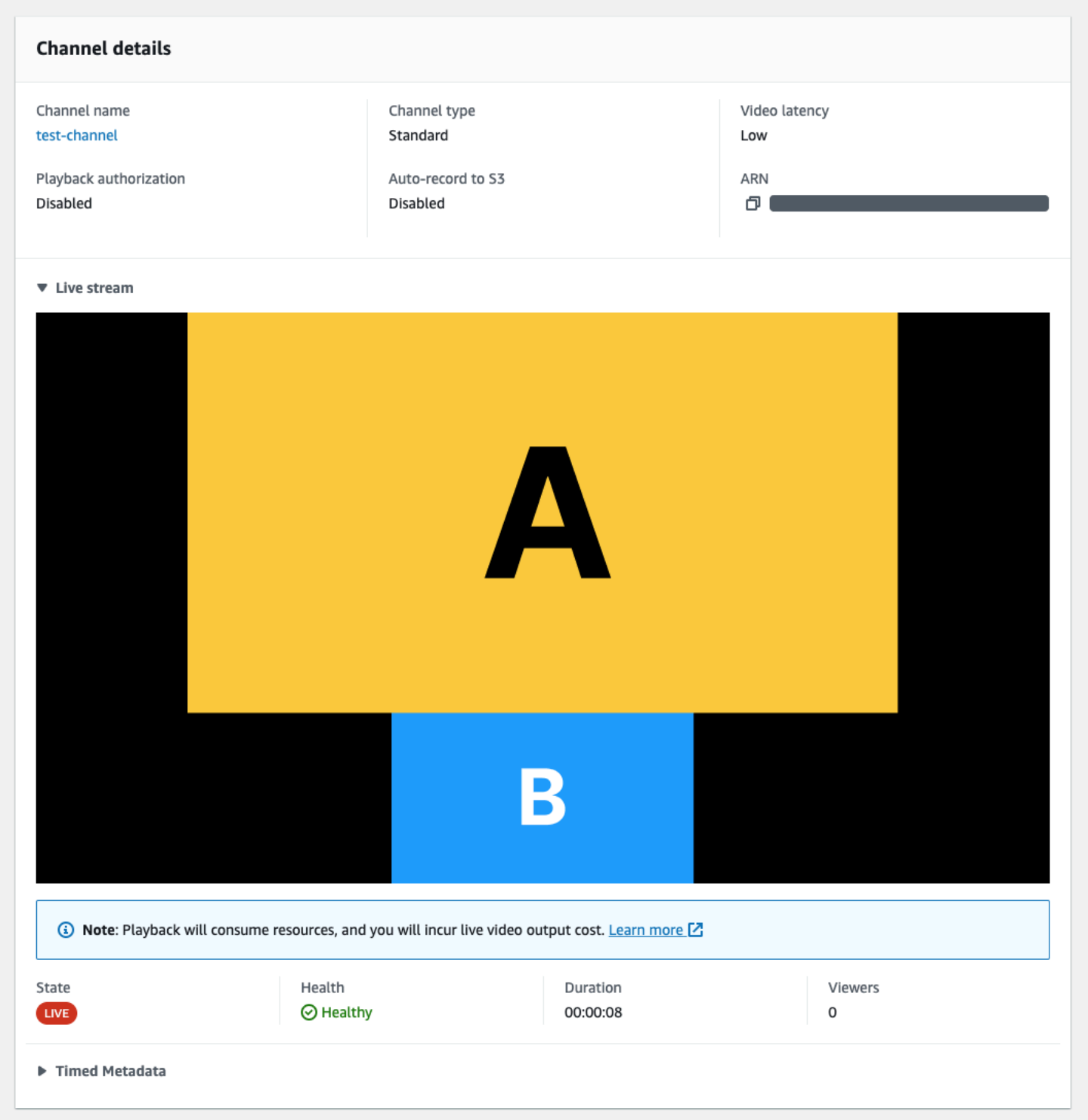
```
aws ivs-realtime start-composition --stage-arn "arn:aws:ivs:us-
east-1:927810967299:stage/8faHz1SQp0ik" --destinations  '[{"channel": {"channelArn":
 "arn:aws:ivs:us-east-1:927810967299:channel/DOlMW4dfMR8r", "encoderConfigurationArn":
 "arn:aws:ivs:us-east-1:927810967299:encoder-configuration/DEkQHWPVaOwO"}}]'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

다음과 같이 응답합니다.

```
{
   "composition" : {
      "arn" : "arn:aws:ivs:us-east-1:927810967299:composition/B19tQcXRgtoz",
      "destinations" : [ {
         "configuration" : {
            "channel" : {
               "channelArn" : "arn:aws:ivs:us-east-1:927810967299:channel/
DOlMW4dfMR8r",
               "encoderConfigurationArn" : "arn:aws:ivs:us-east-1:927810967299:encoder-
configuration/DEkQHWPVaOwO"
            },
            "name" : ""
```

```
         },
         "id" : "SGmgBXTULuXv",
         "state" : "STARTING"
      } ],
      "layout" : {
         "grid" : {
            "featuredParticipantAttribute" : "screen-share"
         }
      },
      "stageArn" : "arn:aws:ivs:us-east-1:927810967299:stage/8faHz1SQp0ik",
      "startTime" : "2023-09-27T21:32:38Z",
      "state" : "STARTING",
      "tags" : { }
   }
}
```

스테이지 E813MFklPWLF 참가자가 스테이지에 참여하면 해당 참가자의 비디오가 추천 슬롯에 표시되고 다른 모든 스테이지 게시자는 슬롯 아래에 렌더링됩니다.



구성 중지

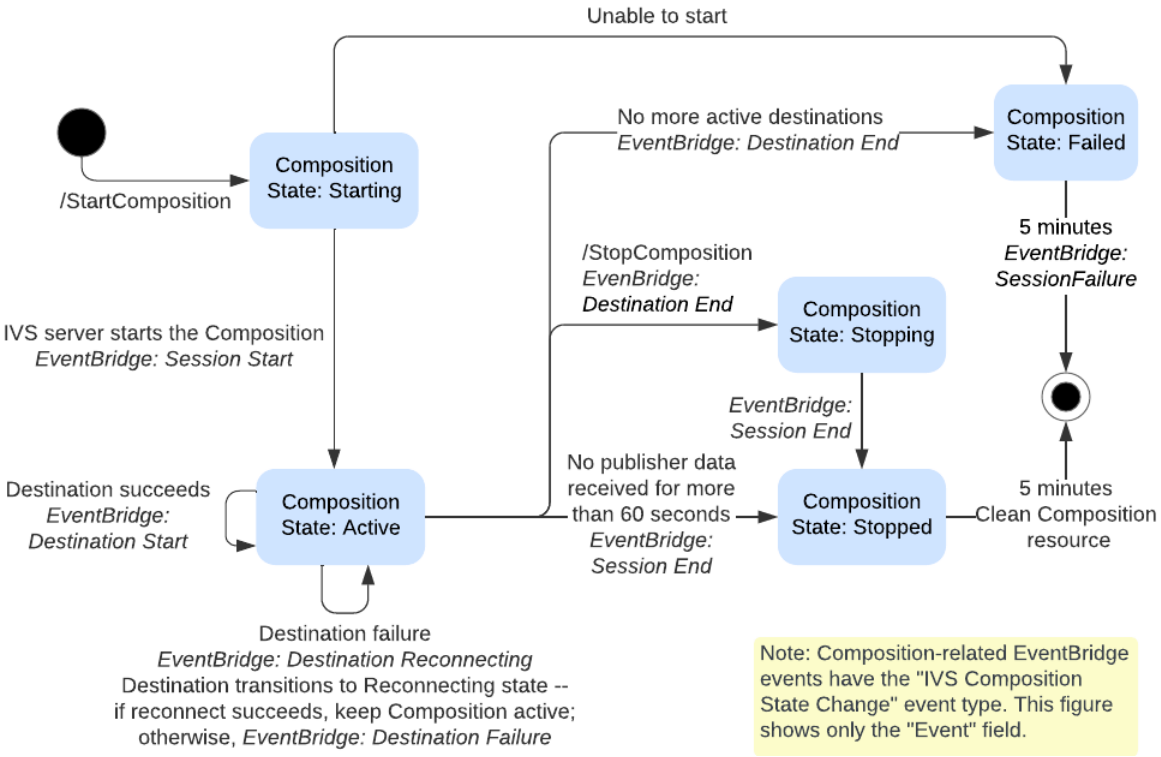
언제든지 구성을 중지하려면 StopComposition 엔드포인트를 호출하세요.

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

구성 수명 주기

아래 다이어그램을 사용하여 구성의 상태 전환을 이해하세요. 대략적으로 구성의 수명 주기는 대략 다음과 같습니다.

1. 구성 리소스는 사용자가 StartComposition 엔드포인트를 호출할 때 생성됩니다.
2. IVS가 구성을 성공적으로 시작하면 "IVS 구성 상태 변경(세션 시작)" EventBridge 이벤트가 전송됩니다. 이벤트에 대한 자세한 내용은 IVS 실시간 스트리밍과 EventBridge 사용을 참조하세요.
3. 구성이 활성 상태가 되면 다음과 같은 일이 발생할 수 있습니다.
   - 사용자가 구성 중지 - StopComposition 엔드포인트가 호출되면 IVS는 구성의 정상 종료를 시작하여 "대상 종료" 이벤트와 "세션 종료" 이벤트를 차례로 전송합니다.
   - 구성이 자동 종료됩니다. - IVS 스테이지에 적극적으로 게시하는 참가자가 없는 경우 60초 후에 구성이 자동으로 완료되고 EventBridge 이벤트가 전송됩니다.
   - 대상 오류 - 대상이 예기치 않게 실패하는 경우(예: IVS 채널 삭제) 대상이 해당 `RECONNECTING` 상태로 전환되고 “대상 재연결” 이벤트가 전송됩니다. 복구가 불가능한 경우 IVS는 대상을 해당 `FAILED` 상태로 전환하고 “대상 실패” 이벤트가 전송됩니다. IVS는 대상 중 하나 이상이 활성 상태인 경우 구성을 활성 상태로 유지합니다.
4. 구성이 `STOPPED` 또는 `FAILED` 상태가 되면 5분 후에 자동으로 정리됩니다. (그러면 ListCompositions이나 GetComposition으로 더 이상 검색되지 않습니다.)


Unable to start
/StartComposition
Composition
State: Starting
No more active destinations
EventBridge: Destination End
Composition
State: Failed
5 minutes
EventBridge:
SessionFailure
/StopComposition
EvenBridge:
Destination End
Composition
State: Stopping
IVS server starts the Composition
EventBridge: Session Start
EventBridge:
Session End
Destination succeeds
EventBridge:
Destination Start
Composition
State: Active
No publisher data
received for more
than 60 seconds
EventBridge:
Session End
Composition
State: Stopped
5 minutes
Clean Composition
resource
Destination failure
EventBridge: Destination Reconnecting
Destination transitions to Reconnecting state --
if reconnect succeeds, keep Composition active;
otherwise, EventBridge: Destination Failure
Note: Composition-related EventBridge
events have the "IVS Composition
State Change" event type. This figure
shows only the "Event" field.

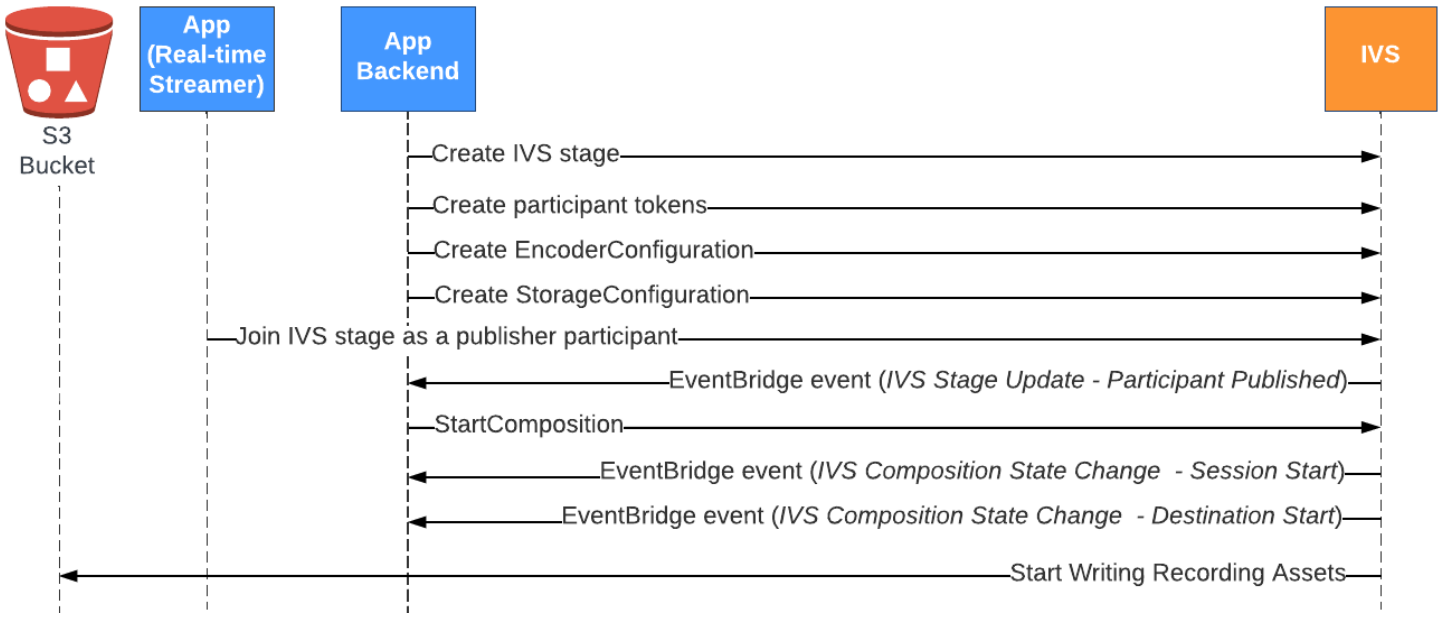
복합 레코딩(실시간 스트리밍)

이 문서에서는 서버 측 구성 내에서 복합 레코딩 기능을 사용하는 방법을 설명합니다. 복합 레코딩을 사용하면 IVS 서버를 사용하여 하나의 보기로 모든 스테이지 게시자를 효과적으로 결합한 다음 결과 비디오를 S3 버킷에 저장함으로써 IVS 스테이지의 HLS 레코딩을 생성할 수 있습니다.

사전 조건

복합 녹화를 사용하려면 활성 게시자가 있는 스테이지와 녹화 대상으로 사용할 S3 버킷이 있어야 합니다. 아래에서는 EventBridge 이벤트를 사용하여 S3 버킷에 컴포지션을 기록할 수 있는 한 가지 워크플로에 대해 설명합니다. 그 대신 자체 앱 로직에 따라 구성을 시작하고 중지할 수 있습니다.

1. 각 게시자를 위한 IVS 스테이지와 참가자 토큰을 생성하세요.
2. EncoderConfiguration(녹화된 비디오의 렌더링 방식을 나타내는 객체) 를 생성합니다.
3. S3 버킷과 a StorageConfiguration(녹화 콘텐츠가 저장될 위치) 를 생성합니다.
4. 스테이지에 참여하고 스테이지에 게시하세요.
5. 참가자 게시 EventBridge 이벤트를 수신하면 S3 DestinationConfiguration 객체를 대상으로 하여 StartComposition호출하십시오.
6. 몇 초 후 S3 버킷에 유지되는 HLS 세그먼트를 확인할 수 있습니다.



참고: 스테이지에 있는 게시자 참가자가 60초 동안 활동이 없으면 구성이 자동 종료됩니다. 이때 구성이 종료되고 STOPPED 상태로 전환합니다. STOPPED 상태로 몇 분 지나면 구성을 자동으로 삭제합니다. 자세한 내용은 서버 측 구성의 구성 수명 주기를 참조하세요.

복합 녹화 예: S3 버킷 대상 StartComposition 포함

아래 예는 S3를 컴포지션의 유일한 대상으로 지정하는 StartComposition엔드포인트에 대한 일반적인 호출을 보여줍니다. 구성이 ACTIVE 상태로 전환되면 비디오 세그먼트와 메타데이터가 storageConfiguration 객체가 지정한 S3 버킷에 기록되기 시작합니다. 레이아웃이 다른 구성을 생성하려면 서버 측 구성의 "레이아웃" 및 IVS 실시간 스트리밍 API 참조를 참조하세요.

요청

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

{
   "destinations": [
      {
         "s3": {
            "encoderConfigurationArns": [
              "arn:aws:ivs:ap-northeast-1:927810967299:encoder-configuration/
PAAwglkRtjge"
            ],
            "storageConfigurationArn": "arn:aws:ivs:ap-
northeast-1:927810967299:storage-configuration/ZBcEbgbE24Cq"
         }
      }
   ],
   "idempotencyToken": "db1i782f1g9",
   "stageArn": "arn:aws:ivs:ap-northeast-1:927810967299:stage/WyGkzNFGwiwr"
}
```

응답

```
{
    "composition": {
        "arn": "arn:aws:ivs:ap-northeast-1:927810967299:composition/s2AdaGUbvQgp",
        "destinations": [
            {
                "configuration": {
```

```
                    "name": "",
                    "s3": {
                        "encoderConfigurationArns": [
                            "arn:aws:ivs:ap-northeast-1:927810967299:encoder-
configuration/PAAwglkRtjge"
                        ],
                        "recordingConfiguration": {
                            "format": "HLS"
                        },
                        "storageConfigurationArn": "arn:aws:ivs:ap-
northeast-1:927810967299:storage-configuration/ZBcEbgbE24Cq"
                    }
                },
                "detail": {
                    "s3": {
                        "recordingPrefix": "MNALAcH9j2EJ/s2AdaGUbvQgp/2pBRKrNgX1ff/
composite"
                    }
                },
                "id": "2pBRKrNgX1ff",
                "state": "STARTING"
            }
        ],
        "layout": null,
        "stageArn": "arn:aws:ivs:ap-northeast-1:927810967299:stage/WyGkzNFGwiwr",
        "startTime": "2023-11-01T06:25:37Z",
        "state": "STARTING",
        "tags": {}
    }
}
```

StartComposition 응답에 있는 `recordingPrefix` 필드를 사용하여 레코딩 콘텐츠를 저장할 위치를 결정할 수 있습니다.

레코딩 콘텐츠

컴포지션이 `ACTIVE` 상태로 전환되면 HLS 비디오 세그먼트와 메타데이터 파일이 호출 시 제공된 S3 버킷에 기록되는 것을 볼 수 있습니다. StartComposition 이러한 콘텐츠는 온디맨드 비디오로 후처리 또는 재생에 사용할 수 있습니다.

구성이 활성화되면 “IVS 구성 상태 변경” 이벤트가 생성되며 매니페스트 파일과 비디오 세그먼트가 기록되기까지 약간의 시간이 걸립니다. "IVS 구성 상태 변경(레코딩 종료)" 이벤트를 받은 후에 레코딩된

스트림을 재생하거나 처리하는 것이 좋습니다. 자세한 내용은 IVS 실시간 EventBridge 스트리밍과 함께 사용을 참조하십시오.

다음은 라이브 IVS 세션 레코딩의 샘플 디렉터리 구조 및 콘텐츠입니다.

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

events 폴더에는 레코딩 이벤트에 해당하는 메타데이터 파일이 들어 있습니다. JSON 메타데이터 파일은 레코딩이 시작되거나, 성공적으로 종료되거나, 실패로 종료될 때 생성됩니다.

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

지정된 events 폴더에는 recording-started.json 및 recording-ended.json 또는 recording-failed.json이 포함됩니다.

여기에는 레코딩된 세션 및 출력 형식과 관련된 메타데이터가 포함됩니다. 다음은 JSON 세부 정보입니다.

media 폴더에는 지원되는 미디어 콘텐츠가 있습니다. hls 하위 폴더에는 구성 세션 중에 생성된 모든 미디어 및 매니페스트 파일이 포함되어 있으며, 이는 IVS 플레이어에서 재생할 수 있습니다. HLS 매니페스트는 multivariant.m3u8 폴더에 있습니다.

에 대한 버킷 정책 StorageConfiguration

StorageConfiguration 객체가 생성되면 IVS는 지정된 S3 버킷에 콘텐츠를 쓸 수 있는 액세스 권한을 얻습니다. 이 액세스는 S3 버킷의 정책을 수정하여 부여됩니다. IVS의 액세스를 제거하는 방식으로 버킷 정책을 변경하면 진행 중인 레코딩과 새 레코딩이 실패합니다.

아래 예는 IVS 가 S3 버킷에 기록할 수 있도록 허용하는 S3 버킷 정책을 보여줍니다.

```
{
    "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 메타데이터 파일

이 메타데이터는 JSON 형식입니다. 이는 다음 정보로 구성됩니다.

필드	유형	필수	설명
`stage_arn`	문자열	예	구성의 소스로 사용되는 스테이지의 ARN입니다.
`media`	객체	예	이 레코딩에 사용할 수 있는 미디어 콘텐츠의 열거된 객체를 포함하는 객체입니다. 유효한 값: `"hls"`.
`hls`	객체	예	Apple HLS 형식 출력을 설명하는 열거형 필드입니다.
`duration_ms`	정수	조건	레코딩된 HLS 콘텐츠의 지속 시간(밀리초)입니다. 이는 `recording`

필드	유형	필수	설명
			_status 가 "RECORDING_ENDED" 또는 "RECORDING_ENDED_WITH_FAILURE" 인 경우에만 사용할 수 있습니다. 레코딩이 완료되기 전에 실패한 경우 이 값은 0입니다.
path	문자열	예	HLS 콘텐츠가 저장되는 S3 접두사의 상대 경로입니다.
playlist	문자열	예	HLS 마스터 재생 목록 파일의 이름입니다.
renditions	객체	예	메타데이터 객체의 변환 배열(HLS 변형)입니다. 항상 하나 이상의 변환이 있습니다.
path	문자열	예	이 변환에 대해 HLS 콘텐츠가 저장되는 S3 접두사의 상대 경로입니다.
playlist	문자열	예	이 변환에 대한 미디어 재생 목록 파일의 이름입니다.
resolution_height	int	조건	인코딩된 비디오의 해상도 높이(픽셀)입니다. 이는 변환에 비디오 트랙이 포함된 경우에만 사용할 수 있습니다.
resolution_width	int	조건	인코딩된 비디오의 해상도 폭(픽셀)입니다. 이는 변환에 비디오 트랙이 포함된 경우에만 사용할 수 있습니다.

필드	유형	필수	설명
recording_ended_at	문자열	조건	레코딩이 종료된 RFC 3339 UTC 타임스탬프입니다. 이는 recording_status 가 "RECORDING_ENDED" 또는 "RECORDING_ENDED_WITH_FAILURE" 인 경우에만 사용할 수 있습니다. recording_started_at 과 recording_ended_at 은 이러한 이벤트가 생성될 때의 타임스탬프이며 HLS 비디오 세그먼트 타임스탬프와 정확히 일치하지 않을 수 있습니다. 레코딩 지속 시간을 정확하게 파악하려면 duration_ms 필드를 사용하세요.
recording_started_at	문자열	조건	레코딩이 시작된 RFC 3339 UTC 타임스탬프입니다. recording_status 가 RECORDING_START_FAILED 인 경우에는 이 기능을 사용할 수 없습니다. recording_ended_at 에 대한 위의 참고 사항을 참조하세요.
recording_status	문자열	예	레코딩 상태입니다. 유효한 값: "RECORDING_STARTED" , "RECORDING_ENDED" , "RECORDING_START_FAILED" , "RECORDING_ENDED_WITH_FAILURE" .

필드	유형	필수	설명
recording_status_message	문자열	조건	상태에 대한 설명 정보입니다. 이는 recording_status 가 "RECORDING_ENDED" 또는 "RECORDING_ENDED_WITH_FAILURE" 인 경우에만 사용할 수 있습니다.
version	문자열	예	메타데이터 스키마의 버전입니다.

예: 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
        }
      ]
    }
  }
}
```

예: 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
        }
      ]
    }
  }
}
```

예: 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
        }
      ]
    }
  }
```

```
}
```

프라이빗 버킷에서 레코딩된 콘텐츠 재생

기본적으로 레코딩된 콘텐츠는 프라이빗이므로 직접 S3 URL을 사용하여 재생할 수 없습니다. IVS 플레이어나 다른 플레이어를 사용하여 재생할 HLS 다변량 재생 목록(m3u8 파일)을 열려고 하면 오류가 발생합니다(예: “요청한 리소스에 액세스할 수 있는 권한이 없습니다”). 대신 Amazon CloudFront CDN(콘텐츠 전송 네트워크) 을 사용하여 이러한 파일을 재생할 수 있습니다.

CloudFront 프라이빗 버킷의 콘텐츠를 제공하도록 배포를 구성할 수 있습니다. 일반적으로 이 방법은 에서 제공하는 제어를 우회하여 읽을 수 있는 공개적으로 액세스할 수 있는 버킷을 사용하는 것보다 좋습니다. CloudFront 프라이빗 오리진 버킷에 대한 읽기 권한을 가진 특수 CloudFront 사용자인 OAC(Origin Access Control) 를 생성하여 프라이빗 버킷에서 배포가 제공되도록 설정할 수 있습니다. 배포를 생성한 후 CloudFront 콘솔이나 API를 통해 OAC를 생성할 수 있습니다. Amazon CloudFront 개발자 안내서의 새 원본 액세스 제어 생성을 참조하십시오.

CORS가 활성화된 CloudFront 상태에서 재생 설정

이 예제에서는 개발자가 CORS가 활성화된 상태로 CloudFront 배포를 설정하여 모든 도메인에서 녹화를 재생할 수 있는 방법을 다룹니다. 이는 개발 단계에서 특히 유용하지만 프로덕션 요구 사항에 맞게 아래 예제를 수정할 수 있습니다.

1단계: S3 버킷 생성

레코딩을 저장하는 데 사용할 S3 버킷을 생성합니다. 버킷은 IVS 워크플로에 사용하는 것과 동일한 리전에 있어야 합니다.

버킷에 허용 CORS 정책 추가

1. AWS 콘솔에서 S3 버킷 권한 탭으로 이동합니다.
2. 아래의 CORS 정책을 복사하여 교차 오리진 리소스 공유(CORS)에 붙여넣습니다. 이는 S3 버킷에서 CORS 액세스를 활성화합니다.

```
[
    {
        "AllowedHeaders": [
            "*"
        ],
```

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

2단계: 배포판 만들기 CloudFront

CloudFront 개발자 안내서의 CloudFront 배포판 만들기를 참조하십시오.

AWS 콘솔을 사용하여 다음 정보를 입력합니다.

다음 표시될 수도 있습니다.	다음 항목을 선택하세요.
원본 도메인	이전 단계에서 생성한 S3 버킷
원본 액세스	기본 파라미터를 사용하는 원본 액세스 제어 설정(권장)
기본 캐시 동작: 뷰어 프로토콜 정책	Redirect HTTP to HTTPS
기본 캐시 동작: 허용된 HTTP 메서드	GET, HEAD, OPTIONS
기본 캐시 동작: 캐시 키 및 원본 요청	CachingDisabled 정책
기본 캐시 동작: 원본 요청 정책	CORS-S3Origin
기본 캐시 동작: 응답 헤더 정책	SimpleCORS
웹 애플리케이션 방화벽	보안 보호 활성화

그런 다음 CloudFront 배포본을 저장합니다.

3단계: S3 버킷 정책 설정

1. S3 StorageConfiguration 버킷에 설정한 모든 항목을 삭제합니다. 이는 해당 버킷에 대한 정책을 생성할 때 자동으로 추가된 모든 버킷 정책을 제거합니다.
2. CloudFront 배포로 이동하여 모든 배포 필드가 이전 단계에서 정의된 상태에 있는지 확인하고 버킷 정책을 복사 (정책 복사 버튼 사용) 합니다.
3. S3 버킷으로 이동합니다. 권한 탭에서 버킷 정책 편집을 선택하고 이전 단계에서 복사한 버킷 정책을 붙여넣습니다. 이 단계 이후에는 버킷 정책에 해당 CloudFront 정책이 독점적으로 적용되어야 합니다.
4. S3 버킷을 지정하여 StorageConfiguration a를 생성합니다.

StorageConfiguration 가 생성되면 S3 버킷 정책에 두 개의 항목이 표시됩니다. 하나는 콘텐츠 CloudFront 읽기를 허용하고 다른 하나는 IVS가 콘텐츠를 쓸 수 있도록 허용합니다. IVS 액세스를 포함하는 최종 버킷 정책의 예는 예: IVS 액세스를 포함한 S3 버킷 정책에 CloudFront 나와 있습니다. CloudFront

4단계: 레코딩 재생

CloudFront 배포를 성공적으로 설정하고 버킷 정책을 업데이트한 후에는 IVS 플레이어를 사용하여 녹화를 재생할 수 있어야 합니다.

1. 구성을 성공적으로 시작하고 S3 버킷에 레코딩이 저장되어 있는지 확인하세요.
2. 이 예제의 1단계부터 3단계까지 수행한 후에는 URL을 통해 비디오 파일을 사용할 수 있어야 합니다 CloudFront . CloudFront URL은 Amazon CloudFront 콘솔의 세부 정보 탭에 있는 배포 도메인 이름입니다. 이 키는 다음과 같은 형식입니다.

   ```
   a1b23cdef4ghij.cloudfront.net
   ```

3. CloudFront 배포를 통해 녹화된 비디오를 재생하려면 s3 버킷에서 `multivariant.m3u8` 파일의 객체 키를 찾으십시오. 이 키는 다음과 같은 형식입니다.

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

4. CloudFront URL 끝에 객체 키를 추가합니다. 최종 URL은 다음과 같습니다.

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

5. 이제 IVS 플레이어의 소스 속성에 최종 URL을 추가하여 전체 레코딩을 시청할 수 있습니다. 레코딩된 비디오를 보려면 IVS Player SDK: 웹 가이드의 시작하기에서 데모를 사용할 수 있습니다.

예: IVS 액세스를 포함한 S3 버킷 정책 CloudFront

아래 스니펫은 프라이빗 버킷에 콘텐츠를 읽을 수 있도록 허용하는 CloudFront S3 버킷 정책과 IVS가 버킷에 콘텐츠를 쓸 수 있도록 허용하는 S3 버킷 정책을 보여줍니다. 참고: 아래 코드 조각을 복사하여 자체 버킷에 붙여넣지 마세요. 정책에는 배포와 관련된 ID가 포함되어야 합니다. CloudFront StorageConfiguration

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "CompositeWrite-7eiKaIGkC9DO",
      "Effect": "Allow",
      "Principal": {
        "Service": "ivs-composite.ap-northeast-1.amazonaws.com"
      },
      "Action": [
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
      "Resource": "arn:aws:s3:::eicheane-test-1026-2-ivs-recordings/*",
      "Condition": {
        "StringEquals": {
          "s3:x-amz-acl": "bucket-owner-full-control"
        },
        "Bool": {
          "aws:SecureTransport": "true"
        }
      }
    },
    {
      "Sid": "AllowCloudFrontServicePrincipal",
      "Effect": "Allow",
      "Principal": {
        "Service": "cloudfront.amazonaws.com"
      },
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::eicheane-test-1026-2-ivs-recordings/*",
      "Condition": {
```

```
        "StringEquals": {
          "AWS:SourceArn": "arn:aws:cloudfront::844311324168:distribution/
E1NG4YMW5MN25A"
        }
      }
    }
  ]
}
```

문제 해결

- 구성이 S3 버킷에 기록되지 않음 — S3 버킷과 StorageConfiguration 객체가 동일한 지역에 생성되었는지 확인하십시오. 또한 버킷 정책을 확인하여 IVS가 버킷에 액세스할 수 있는지 확인하십시오. 버킷 정책을 참조하십시오 StorageConfiguration.
- ListCompositions공연할 때 컴포지션을 찾을 수 없어요. 컴포지션은 임시 리소스입니다. 최종 상태로 전환하면 몇 분 후에 자동으로 삭제합니다.
- 내 구성이 자동으로 중지됩니다 - 스테이지에 60초 이상 게시자가 없으면 구성이 자동으로 중지됩니다.

알려진 문제

구성 진행 중에는 복합 레코딩으로 작성된 미디어 재생 목록에 `#EXT-X-PLAYLIST-TYPE:EVENT` 태그가 붙습니다. 구성이 완료되면 태그가 `#EXT-X-PLAYLIST-TYPE:VOD`로 업데이트됩니다. 원활한 재생 환경을 위해 구성이 성공적으로 완료된 후에만 이 재생 목록을 사용하는 것이 좋습니다.

OBS 및 WHIP 지원 (실시간 스트리밍)

이 문서에서는 OBS와 같은 WHIP 호환 인코더를 사용하여 IVS 실시간 스트리밍에 게시하는 방법을 설명합니다. WHIP (WebRTC-HTTP 통합 프로토콜) 는 WebRTC 통합을 표준화하기 위해 개발된 IETF 초안입니다.

WHIP는 OBS와 같은 소프트웨어와의 호환성을 지원하여 IVS 브로드캐스트 SDK를 대체하는 데스크탑 퍼블리싱을 위한 대안을 제공합니다. OBS에 익숙하며 더 섬세한 스트리머는 장면 전환, 오디오 믹싱, 오버레이 그래픽과 같은 고급 프로덕션 특성 때문에 OBS를 선호할 수도 있습니다. 이를 통해 개발자들은 IVS 웹 브로드캐스트 SDK를 사용하여 직접 브라우저 퍼블리싱을 하거나 스트리머가 데스크탑에서 OBS를 사용하여 더 강력한 도구를 사용할 수 있도록 하는 등 다양한 옵션을 이용할 수 있습니다.

또한 WHIP는 IVS 브로드캐스트 SDK를 사용하는 것이 불가능하거나 선호되지 않는 상황에서도 유용합니다. 예를 들어 하드웨어 인코더와 관련된 설정에서는 IVS 브로드캐스트 SDK가 옵션이 아닐 수 있습니다. 하지만 인코더가 WHIP를 지원하는 경우에도 인코더에서 IVS로 직접 게시할 수 있습니다.

OBS 가이드

OBS는 버전 30부터 WHIP를 지원합니다. 시작하려면 OBS v30 이상 버전 (https://obsproject.com/) 을 다운로드하십시오.

WHIP를 통해 OBS를 사용하여 IVS 스테이지에 게시하려면 다음 단계를 따르십시오.

1. 게시 기능이 있는 참가자 토큰을 생성하십시오. WHIP 용어로, 참가자 토큰은 베어러 토큰입니다. 기본적으로 참가자 토큰은 12시간 후에 만료되지만 기간을 최대 14일까지 연장할 수 있습니다.
2. 설정을 클릭합니다. 설정 패널의 스트림 섹션에 있는 서비스 드롭다운에서 WHIP를 선택합니다.
3. 서버의 경우 https://global.whip.live-video.net/ 을 입력합니다.
4. 베어러 토큰의 경우 2단계에서 생성한 참가자 토큰을 입력합니다.
5. 몇 가지 제한 사항을 제외하고 평소와 같이 비디오 설정을 구성하십시오.
   a. IVS 실시간 스트리밍은 8.5Mbps에서 최대 720p 입력을 지원합니다. 이 한도 중 하나를 초과하면 스트림 연결이 끊어집니다.
   b. 출력 패널의 키프레임 간격을 1초 또는 2초로 설정하는 것이 좋습니다. 키프레임 간격을 짧게 설정하면 시청자가 비디오를 더 빨리 재생할 수 있습니다. 또한 지연 시간을 최소화하려면 CPU 사용 사전 설정을 초고속으로 설정하고 튜닝을 지연 시간을 0으로 설정하는 것이 좋습니다.
   c. OBS는 동시 방송을 지원하지 않으므로 비트 전송률을 2.5Mbps 미만으로 유지하는 것이 좋습니다. 이렇게 하면 저대역폭 연결을 사용하는 시청자가 시청할 수 있습니다.

6. 스트리밍 시작을 누릅니다.

Service Quotas(실시간 스트리밍)

다음은 Amazon Interactive Video Service(IVS) 실시간 엔드포인트, 리소스 및 기타 작업에 대한 서비스 할당량 및 한도입니다. 서비스 할당량(한도)은 AWS 계정의 최대 서비스 리소스 또는 작업 수입니다. 즉, 테이블에 별도로 명시되지 않는 한, 이러한 한도는 AWS 계정별로 다르게 적용됩니다. 자세한 내용은 AWS Service Quotas도 참조하세요.

엔드포인트를 사용하여 AWS 서비스에 프로그래밍 방식으로 연결합니다. AWS 서비스 엔드포인트도 참조하세요.

모든 할당량은 리전별로 적용됩니다.

서비스 할당량 증가

조정 가능한 할당량과 관련하여 AWS 콘솔을 통해 비율 증가를 요청할 수 있습니다. 콘솔을 사용하여 서비스 할당량에 대한 정보도 확인합니다.

API 호출 비율 할당량은 조절할 수 없습니다.

API 호출 비율 할당량

엔드포인트 유형	엔드포인트	기본값
구성	`GetComposition`	5TPS
구성	`ListCompositions`	5TPS
구성	`StartComposition`	5TPS
구성	`StopComposition`	5TPS
MediaEncoder	`CreateEncoderConfiguration`	5TPS
MediaEncoder	`DeleteEncoderConfiguration`	5TPS
MediaEncoder	`GetEncoderConfiguration`	5TPS
MediaEncoder	`ListEncoderConfigurations`	5TPS

엔드포인트 유형	엔드포인트	기본값
단계	CreateParticipantToken	50TPS
단계	CreateStage	5TPS
단계	DeleteStage	5TPS
단계	DisconnectParticipant	5TPS
단계	GetParticipant	5TPS
단계	GetStage	5TPS
단계	GetStageSession	5TPS
단계	ListStages	5TPS
단계	UpdateStage	5TPS
단계	ListParticipants	5TPS
단계	ListParticipantEvents	5TPS
단계	ListStageSessions	5TPS
StorageConfiguration	CreateStorageConfiguration	5TPS
StorageConfiguration	DeleteStorageConfiguration	5TPS
StorageConfiguration	GetStorageConfiguration	5TPS
StorageConfiguration	ListStorageConfigurations	5TPS
태그	ListTagsForResource	10TPS
태그	TagResource	10TPS
태그	UntagResource	10TPS

기타 할당량

리소스 또는 기능	기본값	조정 가능	Description
EncoderConfigurations	20	예	계정당 인코더 구성 리소스의 최대 수입니다.
구성 대상	2	아니요	구성 리소스의 최대 대상 개체 수입니다.
구성: 최대 지속 시간	24	아니요	구성이 존재할 수 있는 시간 최대값(시간)입니다.
구성	5	예	계정당 최대 동시 구성 리소스입니다.
참가자 게시 또는 구독 기간	24	아니요	참가자가 스테이지를 게시하거나 스테이지 구독 상태를 유지할 수 있는 최대 시간입니다.
참가자 게시 해상도	720p	아니요	참가자가 게시한 비디오의 최대 해상도입니다.
참가자 다운로드 비트레이트	8.5Mbps	아니요	참가자의 모든 구독에 대한 최대 집계 다운로드 비트레이트입니다.
스테이지 참가자(게시자)	12	아니요	한 번에 스테이지에 게시할 수 있는 최대 참가자 수입니다.
스테이지 참가자(구독자)	10,000개	예	한 번에 스테이지를 구독할 수 있는 최대 참가자 수입니다.
Stages	100	예	AWS 리전당 최대 스테이지 수입니다.

실시간 스트리밍 최적화

IVS 실시간 스트리밍을 사용하여 비디오를 스트리밍하고 볼 때 사용자가 최상의 경험을 할 수 있도록 현재 제공되는 기능을 사용하여 경험의 일부를 개선하거나 최적화할 수 있는 몇 가지 방법이 있습니다.

소개

사용자 경험의 품질을 최적화할 때 사용자가 원하는 환경을 고려하는 것이 중요하며, 이는 사용자가 시청하는 콘텐츠와 네트워크 상태에 따라 달라질 수 있습니다.

이 가이드에서는 스트림 게시자 또는 스트림 구독자인 사용자에게 초점을 맞추고 해당 사용자가 원하는 작업과 경험을 고려합니다.

적응형 스트리밍: 동시 방송을 사용한 계층화된 인코딩

이 기능은 다음 클라이언트 버전에서만 지원됩니다.

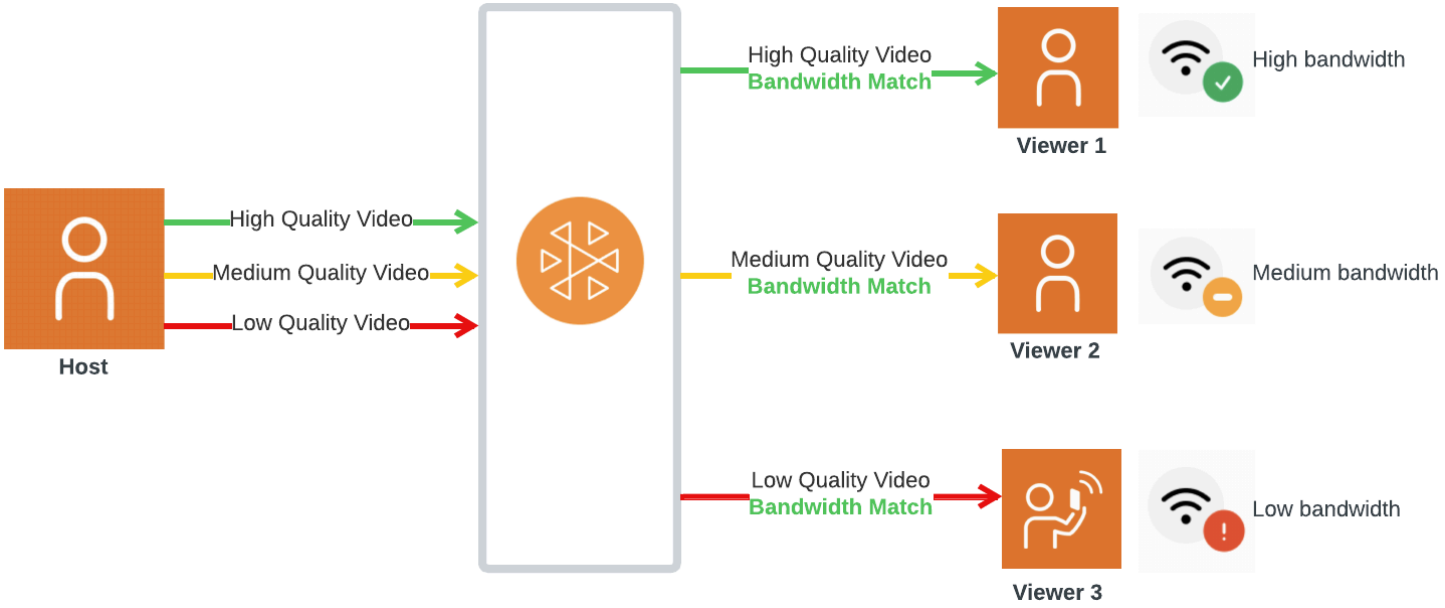
- iOS 및 Android 1.12.0+
- 웹 1.5.1+

계정에서 이 기능을 사용하려면 amazon-ivs-simulcast@amazon .com으로 이메일을 보내야 합니다. SDK 구성을 통해 동시 방송을 활성화해도 옵트인하지 않는 한 효과가 없습니다.

해당 기능을 선택한 후 IVS 실시간 브로드캐스트 SDK를 사용할 때 게시자는 비디오의 여러 계층을 인코딩하고 구독자는 네트워크에 가장 적합한 품질로 자동으로 조정하거나 변경합니다. 이를 동시 방송을 사용한 계층화된 인코딩이라고 합니다.

동시 방송을 사용한 계층화된 인코딩은 Android 및 iOS와 Chrome 데스크톱 브라우저(Windows 및 macOS용)에서 지원됩니다. 다른 브라우저에서는 계층화된 인코딩을 지원하지 않습니다.

아래 다이어그램에서 호스트는 3가지 비디오 품질(높음, 중간, 낮음)을 전송하고 있습니다. IVS는 사용 가능한 대역폭에 따라 각 시청자에게 최고 품질의 비디오를 전달합니다. 이는 각 시청자에게 최적의 경험을 제공합니다. 시청자 1의 네트워크 연결이 양호에서 불량으로 변경되면 IVS는 자동으로 시청자 1에게 더 낮은 품질의 비디오를 전송하기 시작하므로, 시청자 1은 가능한 최상의 품질로 스트림을 중단 없이 계속 시청할 수 있습니다.



기본 계층, 품질 및 프레임 속도

모바일 및 웹 사용자에게 제공되는 기본 품질과 계층은 다음과 같습니다.

모바일(Android, iOS)	웹(Chrome)
상위 계층 또는 사용자 지정: • 최대 비트레이트: 90만 bps • 프레임 속도: 15fps • 해상도: 360x640	상위 계층 또는 사용자 지정: • 최대 비트레이트: 170만 bps • 프레임 속도: 30fps • 해상도: 1280x720
중간 계층: 없음(모바일에서 상위 계층 비트레이트와 하위 계층 비트레이트의 차이가 적기 때문에 필요 없음)	중간 계층: • 최대 비트레이트: 70만 bps • 프레임 속도: 20fps • 해상도: 640x360
하위 계층: • 최대 비트레이트: 15만 bps • 프레임 속도: 15fps • 해상도: 180x320	하위 계층: • 최대 비트레이트: 20만 bps • 프레임 속도: 15fps • 해상도: 320x180

동시 방송을 사용한 계층화된 인코딩 구성

동시 방송에서 레이어 인코딩을 사용하려면 이 기능을 선택하고 클라이언트에서 이 기능을 활성화해야 합니다. 이 기능을 활성화하면 비디오 정지가 줄어들어 전체 전송률이 증가하는 것을 확인할 수 있습니다.

Android

```
// Opt-out of Simulcast
StageVideoConfiguration config = new StageVideoConfiguration();
config.simulcast.setEnabled(true);

ImageLocalStageStream cameraStream = new ImageLocalStageStream(frontCamera, config);

// Other Stage implementation code
```

iOS

```
// Opt-out of Simulcast
let config = IVSLocalStageStreamVideoConfiguration()
config.simulcast.enabled = true

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

// Other Stage implementation code
```

웹

```
// Opt-out of Simulcast
let cameraStream = new LocalStageStream(cameraDevice, {
   simulcast: { enabled: true }
})

// Other Stage implementation code
```

스트리밍 구성

이 섹션에서는 비디오 및 오디오 스트림에 사용할 수 있는 기타 구성에 대해 알아봅니다.

스트림 비디오 비트레이트 변경

비디오 스트림의 비트레이트를 변경하려면 다음 구성 샘플을 사용합니다.

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
```

웹

```
// Note: On web it is also recommended to configure the framerate of your device from
 userMedia
const camera = await navigator.mediaDevices.getUserMedia({
   video: {
      bitrate: {
         ideal: 1500,
         max: 1500,
      },
   },
});

let cameraStream = new LocalStageStream(camera.getVideoTracks()[0], {
   // Update Max Bitrate to 1.5mbps or 1500kbps
   maxBitrate: 1500
```

```
})

// Other Stage implementation code
```

비디오 스트림 프레임 속도 변경

비디오 스트림의 프레임 속도를 변경하려면 다음 구성 샘플을 사용합니다.

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
```

웹

```
// 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
```

오디오 비트레이트 및 스테레오 지원 최적화

오디오 스트림의 비트레이트 및 스테레오 설정을 변경하려면 다음 구성 샘플을 사용합니다.

웹

```
// 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
```

권장 최적화

시나리오	추천
프레젠테이션이나 슬라이드와 같이 텍스트나 느리게 움직이는 콘텐츠가 포함된 스트림	동시 방송을 사용한 계층화된 인코딩을 사용하거나 프레임 속도가 낮은 스트림을 구성할 수 있습니다.
동작이 많거나 움직임이 많은 스트림	동시 방송을 사용한 계층화된 인코딩을 사용하세요.
대화가 있거나 움직임이 적은 스트림	동시 방송을 사용한 계층화된 인코딩을 사용하거나 오디오 전용(실시간 스트리밍 브로드캐스트 SDK 가이드: 웹, Android, iOS의 "참가자 구독" 참조)을 선택하세요.
제한된 데이터로 스트리밍하는 사용자	동시 방송을 사용한 계층화된 인코딩을 사용하거나, 모든 사용자의 데이터 사용량을 줄이려면 프레임 속도를 낮추고, 비트레이트를 수동으로 낮추세요.

리소스 및 지원(실시간 스트리밍)

리소스

https://ivs.rocks/는 게시된 콘텐츠(데모, 코드 샘플, 블로그 게시물)를 검색하고, 비용을 예측하고, 라이브 데모를 통해 Amazon IVS를 경험할 수 있는 전용 사이트입니다.

데모

iOS 및 Android용 IVS 실시간 스트리밍 데모는 개발자에게 Amazon IVS를 사용하여 소셜 사용자가 생성한 매력적인 실시간 콘텐츠 애플리케이션을 구축하는 방법을 보여줍니다. 이 애플리케이션은 사용자가 생성한 실시간 스트림의 스크롤 가능한 피드를 제공합니다. 사용자는 비디오 스트림과 오디오 전용 룸을 생성할 수 있습니다. 비디오 스트림 게스트는 게스트 스팟 또는 비교(VS) 모드로 참가할 수 있습니다. 필요한 백엔드를 배포하고 애플리케이션을 구축하는 방법에 대한 지침은 다음 GitHub 리포지토리에서 확인할 수 있습니다.

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

- Android: https://github.com/aws-samples/amazon-ivs-real-time-for-android-demo/
- 백엔드: https://github.com/aws-samples/amazon-ivs-real-time-serverless-demo/

지원

AWS Support Center는 AWS 솔루션을 지원하는 도구 및 전문 지식에 액세스할 수 있도록 다양한 플랜을 제공합니다. 모든 지원 플랜은 연중무휴 24시간 고객 서비스를 제공합니다. AWS 환경을 계획, 배포 및 개선하기 위한 기술 지원과 더 많은 리소스를 받으려면 AWS 사용 사례에 가장 적합한 지원 플랜을 선택합니다.

AWS 프리미엄 서포트는 AWS에서 애플리케이션을 구축 및 실행하도록 지원하기 위해 신속한 응답을 제공하는 1:1 지원 채널입니다.

AWS re:Post는 개발자가 Amazon IVS 관련 기술적 문제에 대해 토론할 수 있는 커뮤니티 기반 Q&A 사이트입니다.

문의처에는 결제 또는 계정과 관련한 비기술적 문의를 보낼 수 있는 링크가 있습니다. 기술적인 질문의 경우 토론 포럼이나 위의 지원 링크를 이용하십시오.

용어집

AWS 용어집도 참조하세요. 아래 표에서 LL은 IVS 짧은 지연 시간 스트리밍을 나타내고, RT는 IVS 실시간 스트리밍을 나타냅니다.

용어	설명	LL	RT	채팅
AAC	고급 오디오 코딩입니다. AAC는 손실이 발생하는 디지털 오디오 압축을 위한 오디오 코딩 표준입니다. MP3의 후속 형식으로 설계된 AAC는 일반적으로 동일한 비트레이트에서 MP3보다 음질이 우수합니다. AAC는 ISO와 IEC에서 MPEG-2 및 MPEG-4 사양의 일부로 표준화됩니다.	✓	✓	
적응형 비트레이트 스트리밍	적응형 비트레이트(ABR) 스트리밍에서는 IVS 플레이어가 연결 품질 저하 시 더 낮은 비트레이트로 전환되고, 연결 품질 향상 시 더 높은 비트레이트로 전환될 수 있습니다.	✓		
적응형 스트리밍	동시 방송을 사용한 계층화된 인코딩을 참조하세요.		✓	
관리 사용자	AWS 계정에서 사용할 수 있는 리소스 및 서비스에 대한 관리 액세스 권한이 있는 AWS 사용자입니다. AWS 설정 사용 설명서의 용어를 참조하세요.	✓	✓	✓
ARN	AWS 리소스의 고유한 식별자인 Amazon 리소스 이름입니다. 구체적인 ARN 형식은 리소스에 따라 다릅니다. IVS 리소스에서 사용하는 ARN 형식은 서비스 승인 참조에서 확인하세요.	✓	✓	✓
가로 세로 비율	프레임 너비와 프레임 높이의 비율을 설명합니다. 예를 들면 16:9는 Full HD 또는 1080p 해상도에 해당하는 종횡비입니다.	✓	✓	
오디오 모드	다양한 유형의 모바일 디바이스 사용자 및 해당 사용자가 사용하는 장비에 최적화된 사전 설정 또는 사용자 지정 오디오 구성입니다. IVS 브로드캐스트		✓	

용어	설명	LL	RT	채팅
	SDK: 모바일 오디오 모드(실시간 스트리밍)를 참조하세요.			
AVC, H.264, MPEG-4 Part 10	H.264 또는 MPEG-4 Part 10이라고도 하는 고급 비디오 코딩은 손실이 발생하는 디지털 비디오 압축을 위한 비디오 압축 표준입니다.	✓	✓	
배경 교체	라이브 스트림 생성자가 배경을 변경할 수 있는 카메라 필터의 일종입니다. IVS 브로드캐스트 SDK: 타사 카메라 필터(실시간 스트리밍)의 배경 교체를 참조하세요.		✓	
비트 전송률	전송되거나 수신되는 초당 비트 수에 대한 스트리밍 지표입니다.	✓	✓	
브로드캐스트, 브로드캐스터	스트림, 스트리머에 대한 다른 용어입니다.	✓		
버퍼링	재생 디바이스에서 재생해야 하는 콘텐츠를 다운로드할 수 없을 때 발생하는 상태입니다. 버퍼링은 여러 가지 방식으로 나타날 수 있습니다. 즉, 콘텐츠가 무작위로 중지 및 시작되거나(끊김이라고도 함) 콘텐츠가 오랫동안 중지되거나(멈춤이라고도 함) IVS 플레이어에서 재생이 일시 중지될 수 있습니다.	✓	✓	
바이트 범위 재생 목록	표준 HLS 재생 목록보다 세분화된 재생 목록입니다. 표준 HLS 재생 목록은 10초짜리 미디어 파일로 구성됩니다. 바이트 범위 재생 목록의 경우 세그먼트 지속 시간은 스트림에 구성된 키프레임 간격과 동일합니다. 바이트 범위 재생 목록은 S3 버킷에 자동 레코딩된 브로드캐스트에만 사용할 수 있습니다. HLS 재생 목록과 함께 생성됩니다. Amazon S3에 자동 레코딩(저지연 스트리밍)의 바이트 범위 재생 목록을 참조하세요.	✓		

용어	설명	LL	RT	채팅
CBR	고정 비트레이트는 브로드캐스트 중에 발생하는 상황과 관계없이 전체 비디오 재생에서 일관된 비트레이트를 유지하는 인코더의 속도 제어 방법입니다. 원하는 비트레이트를 달성하기 위해 작업의 빈 곳을 채울 수 있으며, 대상 비트레이트와 일치하도록 인코딩 품질을 조정하여 피크를 양자화할 수 있습니다. VBR 대신에 CBR을 사용하는 것이 좋습니다.	✓	✓	
CDN	스트리밍 비디오와 같은 콘텐츠를 사용자가 있는 곳으로 가까이 가져와서 전송을 최적화하는 지리적으로 분산된 솔루션인 콘텐츠 전송 네트워크 또는 콘텐츠 배포 네트워크입니다.	✓		
Channel	수집 서버, 스트림 키, 재생 URL 및 레코딩 옵션을 포함하여 스트리밍의 구성을 저장하는 IVS 리소스입니다. 스트리머는 채널과 연결된 스트림 키를 사용하여 브로드캐스트를 시작합니다. 브로드캐스트 중 생성된 모든 지표 및 이벤트는 채널 리소스와 연결됩니다.	✓		
채널 유형	채널에 허용되는 해상도와 프레임 속도를 결정합니다. IVS Low-Latency Streaming API Reference의 채널 유형을 참조하세요.	✓		
챗 로깅	로깅 구성을 채팅룸과 연결하여 활성화할 수 있는 고급 옵션입니다.			✓
채팅룸	메시지 검토 핸들러 및 챗 로깅과 같은 선택적 특성을 포함하여 채팅 세션의 구성을 저장하는 IVS 리소스입니다. IVS 챗 시작하기의 Step 2: Create a Chat Room을 참조하세요.			✓

용어	설명	LL	RT	채팅
클라이언트 측 구성	호스트 디바이스를 사용하여 스테이지 참가자의 오디오 및 비디오 스트림을 혼합한 다음 복합 스트림으로 IVS 채널에 보냅니다. 그러면 클라이언트 리소스 사용률이 높아지고 시청자에게 영향을 미치는 스테이지 또는 호스트 문제가 발생할 위험이 커지는 대신에 구성의 모양을 더 잘 제어할 수 있습니다. 서버 측 구성도 참조하세요.	✓	✓	
CloudFront	아마존에서 제공하는 CDN 서비스입니다.	✓		
CloudTrail	AWS와 외부 소스의 이벤트 및 계정 활동을 수집, 모니터링, 분석 및 유지하는 AWS 서비스입니다. CloudTrailAWS를 사용한 IVS API 호출 로깅을 참조하십시오.	✓	✓	✓
CloudWatch	애플리케이션을 모니터링하고, 성능 변화에 대응하고, 리소스 사용을 최적화하고, 운영 상태에 대한 인사이트를 제공하는 AWS 서비스입니다. IVS 지표를 모니터링하는 CloudWatch 데 사용할 수 있습니다. IVS 실시간 스트리밍 모니터링 및 IVS 지연 시간이 짧은 스트리밍 모니터링을 참조하십시오.	✓	✓	✓
구성	여러 소스의 오디오 및 비디오 스트림을 단일 스트림으로 결합하는 프로세스입니다.	✓	✓	
구성 파이프라인	여러 스트림을 결합하고 결과 스트림을 인코딩하는 데 필요한 일련의 프로세스 단계입니다.	✓	✓	
압축	원래 표현보다 적은 비트를 사용하는 정보 인코딩입니다. 모든 특정 압축은 무손실이거나 손실 허용입니다. 무손실 압축에서는 통계적 중복성을 식별하고 제거하여 비트를 줄입니다. 무손실 압축에서는 손실되는 정보가 없습니다. 손실 허용 압축에서는 불필요하거나 덜 중요한 정보를 제거하여 비트를 줄입니다.	✓	✓	

용어	설명	LL	RT	채팅
컨트롤 플레인	채널, 스테이지 또는 채팅룸과 같은 IVS 리소스에 대한 정보를 저장하고 이러한 리소스를 생성 및 관리하는 인터페이스를 제공합니다. 리전에 따라 다릅니다(AWS 리전 기준).	✓	✓	✓
CORS	한 도메인에서 로드된 클라이언트 웹 애플리케이션이 다른 도메인의 S3 버킷과 같은 리소스와 상호작용하도록 허용하는 AWS 특성인 교차 오리진 리소스 공유(CORS)입니다. 헤더, HTTP 메서드 및 원본 도메인을 기반으로 액세스를 구성할 수 있습니다. Amazon Simple Storage Service 사용 설명서의 교차 오리진 리소스 공유(CORS) 사용 - Amazon Simple Storage Service를 참조하세요.	✓		
사용자 지정 이미지 소스	사전 설정된 카메라로 제한되지 않고 애플리케이션에서 자체 이미지 입력을 제공하도록 IVS 브로드캐스트 SDK에서 제공하는 인터페이스입니다.	✓	✓	
데이터 영역	수집에서 송신까지 데이터를 전달하는 인프라입니다. 컨트롤 플레인에서 관리되는 구성을 기반으로 작동하며 AWS 리전에만 국한되지 않습니다.	✓	✓	✓
인코더, 인코딩	비디오 및 오디오 콘텐츠를 스트리밍에 적합한 디지털 형식으로 변환하는 프로세스입니다. 인코딩은 하드웨어 또는 소프트웨어 기반일 수 있습니다.	✓	✓	
Event	IVS에서 모니터링 서비스에 게시하는 자동 알림입니다. AmazonEventBridge 이벤트는 스테이지 또는 구성 파이프라인과 같은 스트리밍 리소스의 상태 변경을 나타냅니다. 지연 시간이 짧은 IVS EventBridge 스트리밍과 함께 Amazon 사용하기 및 IVS 실시간 스트리밍과 EventBridge 함께 Amazon 사용을 참조하십시오.	✓	✓	✓

용어	설명	LL	RT	채팅
FFmpeg	비디오 및 오디오 파일과 스트림을 처리하는 라이브러리 및 프로그램 모음으로 구성된 무료 오픈 소스 소프트웨어 프로젝트입니다. FFmpeg에서는 오디오와 비디오를 녹음, 변환 및 스트리밍하는 교차 플랫폼 솔루션이 제공됩니다.	✓		
조각화된 스트림	채널의 레코딩 구성에 지정된 간격 내에 브로드캐스트의 연결이 해제되었다가 다시 연결될 때 생성됩니다. 여러 개의 결과 스트림이 단일 브로드캐스트로 간주되며 레코딩된 단일 스트림으로 병합됩니다. Amazon S3에 자동 레코딩(저지연 스트리밍)의 조각화된 스트림 병합을 참조하세요.	✓		
프레임 속도	전송되거나 수신되는 초당 비디오 프레임 수에 대한 스트리밍 지표입니다.	✓	✓	
HLS	IVS 스트림을 시청자에게 전송하는 데 사용되는 HTTP 기반 가변 비트레이트 스트리밍 통신 프로토콜인 HLS(HTTP Live Streaming)입니다.	✓		
HLS 재생 목록	스트림을 구성하는 미디어 세그먼트 목록입니다. 표준 HLS 재생 목록은 10초짜리 미디어 파일로 구성됩니다. HLS에서는 더 세분화된 바이트 범위 재생 목록도 지원합니다.	✓		
Host	비디오 및/또는 오디오를 스테이지로 보내는 실시간 이벤트 참가자입니다.		✓	
IAM	사용자가 자격 증명을 안전하게 관리하고 IVS를 포함한 AWS 서비스 및 리소스에 액세스할 수 있게 하는 AWS 서비스인 Identity and Access Management입니다.	✓	✓	✓
수집	호스트 또는 방송사로부터 비디오 스트림을 수신하여 처리하거나 시청자 또는 다른 참가자에게 전송하는 IVS 프로세스입니다.	✓	✓	

용어	설명	LL	RT	채팅
수집 서버	시청자에게 전송하기 위해 비디오 스트림을 수신하고 트랜스코딩 시스템으로 전송하여 HLS로 트랜스먹싱하거나 트랜스코딩합니다. 수집 서버는 수집 프로토콜(RTMP, RTMPS)과 함께 채널용 스트림을 수신하는 특정 IVS 구성 요소입니다. IVS 지연 시간이 짧은 스트리밍 시작하기의 채널 생성에 대한 정보를 참조하세요.		✓	
인터레이스 비디오	후속 프레임의 홀수 또는 짝수 라인만 전송하고 표시하여 추가 대역폭을 소비하지 않고 두 배로 인식되는 프레임 속도를 생성합니다. 비디오 품질 문제 때문에 인터레이스 비디오는 사용하지 않는 것이 좋습니다.	✓	✓	
JSON	JavaScript 객체 표기법은 사람이 읽을 수 있는 텍스트를 사용하여 속성-값 쌍과 배열 데이터 유형 또는 기타 직렬화 가능한 값으로 구성된 데이터 객체를 전송하는 개방형 표준 파일 형식입니다.	✓	✓	✓
키프레임, 델타 프레임, 키프레임 간격	키프레임(인트라 코딩된 프레임 또는 i-프레임이라고도 함)은 비디오 이미지의 전체 프레임입니다. 후속 프레임인 델타 프레임(예측된 프레임 또는 p-프레임이라고도 함)에는 변경된 정보만 포함됩니다. 인코더에 정의된 키프레임 간격에 따라 스트림 내에서 키프레임이 여러 번 나타납니다.	✓	✓	
Lambda	서버 인프라를 프로비저닝하지 않고 코드(Lambda 함수라고 함)를 실행하는 AWS 서비스입니다. Lambda 함수는 이벤트 및 간접 호출 요청에 대한 응답으로 또는 일정에 따라 실행될 수 있습니다. 예를 들어 IVS 챗에서는 Lambda 함수를 사용하여 채팅룸에 대한 메시지 검토를 활성화합니다.	✓	✓	✓

용어	설명	LL	RT	채팅
지연 시간 glass-to-glass, 지연 시간	데이터 전송에서 발생하는 지연입니다. IVS에서는 지연 시간 범위를 다음과 같이 정의합니다. • 짧은 지연 시간: 3초 미만 • 실시간 지연 시간: 300ms 미만 G lass-to-glass 지연 시간은 카메라가 실시간 스트림을 캡처한 시점부터 시청자 화면에 스트림이 나타나는 시점까지의 지연을 말합니다.	✓	✓	
동시 방송을 사용한 계층화된 인코딩	품질 수준이 각기 다른 여러 비디오 스트림의 동시 인코딩 및 게시를 활성화합니다. 실시간 스트리밍 최적화의 적응형 스트리밍: 동시 방송을 사용한 계층화된 인코딩을 참조하세요.		✓	
메시지 검토 핸들러	채팅룸으로 전송되기 전에 IVS 챗 고객이 사용자 채팅 메시지를 자동으로 검토/필터링할 수 있도록 합니다. Lambda 함수를 채팅룸과 연결하면 활성화됩니다. Chat Message Review Handler의 Creating a Lambda Function을 참조하세요.			✓
믹서	여러 오디오 및 비디오 소스를 이용하여 단일 출력을 생성하는 IVS 모바일 브로드캐스트 SDK의 특성입니다. 카메라, 마이크, 화면 캡처, 애플리케이션에서 생성된 오디오 및 비디오 등의 소스를 나타내는 화면의 비디오 및 오디오 요소 관리를 지원합니다. 해당 출력은 IVS로 스트리밍될 수 있습니다. IVS 브로드캐스트 SDK: 믹서 가이드(지연 시간이 짧은 스트리밍)의 믹싱을 위한 브로드캐스트 세션 구성을 참조하세요.	✓		

용어	설명	LL	RT	채팅
다중 호스트 스트리밍	다중 호스트의 스트림을 단일 스트림으로 결합합니다. 이 작업은 클라이언트 측 또는 서버 측 구성을 사용하여 수행할 수 있습니다. 다중 호스트 스트리밍에서는 Q&A 스테이지로 시청자 초대, 호스트 간 경쟁, 영상 채팅, 대규모 인원 앞에서 서로 대화하는 호스트와 같은 시나리오를 지원합니다.		✓	
다변량 재생 목록	브로드캐스트에 사용할 수 있는 모든 변형 스트림의 인덱스입니다.	✓		
OAC	Origin Access Control은 녹화된 스트림과 같은 콘텐츠가 CloudFrontCDN을 통해서만 제공되도록 S3 버킷에 대한 액세스를 제한하는 메커니즘입니다.	✓		
OBS	Open Broadcaster Software(OBS)는 비디오 레코딩 및 라이브 스트리밍을 위한 무료 오픈 소스 소프트웨어입니다. OBS에서는 데스크톱 게시에 대한 대안을 IVS 브로드캐스트 SDK에 제공합니다. OBS에 익숙하며 더 섬세한 스트리머는 장면 전환, 오디오 믹싱, 오버레이 그래픽과 같은 고급 프로덕션 특성 때문에 OBS를 선호할 수도 있습니다.	✓	✓	
Participant	호스트 또는 시청자로 스테이지에 연결된 실시간 사용자입니다.		✓	
참가자 토큰	실시간 이벤트 참가자가 스테이지에 조인할 때 인증합니다. 참가자 토큰은 스테이지에 대한 참가자의 비디오 전송 가능 여부도 제어합니다.		✓	

용어	설명	LL	RT	채팅
재생 토큰, 재생 키 페어	고객이 프라이빗 채널의 비디오 재생을 제한할 수 있는 인증 메커니즘입니다. 재생 토큰은 재생 키 페어에서 생성됩니다. 재생 키 페어는 재생을 위해 시청자 권한 부여 토큰에 서명하고 이를 검증하는 데 사용되는 퍼블릭-프라이빗 키 페어입니다. 프라이빗 채널 설정의 재생 키 생성 또는 가져오기를 참조하고 IVS Low-Latency API Reference의 Playback Key Pair endpoints를 참조하세요.	✓		
재생 URL	시청자가 특정 채널의 재생을 시작하는 데 사용하는 주소를 식별합니다. 이 주소는 글로벌로 사용할 수 있습니다. IVS에서는 각 시청자에게 비디오를 전송하기 위해 IVS 글로벌 콘텐츠 전송 네트워크에서 최적의 위치를 자동으로 선택합니다. IVS 지연 시간이 짧은 스트리밍 시작하기의 채널 생성에 대한 정보를 참조하세요.	✓		
프라이빗 채널	고객이 재생 토큰 기반의 인증 메커니즘을 사용하여 스트림에 대한 액세스를 제한할 수 있습니다. 프라이빗 채널 설정의 프라이빗 채널 워크플로를 참조하세요.	✓		
프로그레시브 비디오	각 프레임의 모든 라인을 순서대로 전송하고 표시합니다. 브로드캐스트의 모든 스테이지에서 프로그레시브 비디오를 사용하는 것이 좋습니다.	✓	✓	
할당량	AWS 계정의 최대 IVS 서비스 리소스 또는 작업 수입니다. 즉, 별도로 명시되지 않는 한 이러한 한도는 AWS 계정별로 다르게 적용됩니다. 모든 할당량은 리전별로 적용됩니다. AWS 일반 참조 안내서의 Amazon Interactive Video Service 엔드포인트 및 할당량을 참조하세요.	✓	✓	✓

용어	설명	LL	RT	채팅
리전	특정 지리적 영역에 물리적으로 상주하는 AWS 서비스에 대한 액세스를 제공합니다. 리전에서는 내결함성, 안정성 및 복원성을 지원하고 지연 시간을 줄일 수도 있습니다. 리전을 통해 사용자는 가용 상태를 유지하며 리전 중단의 영향을 받지 않는 중복 리소스를 생성할 수 있습니다. 대부분의 AWS 서비스 요청은 특정한 지리적 리전과 관련이 있습니다. 한 리전에서 생성한 리소스는 AWS 서비스에서 제공하는 복제 특성을 명시적으로 사용하지 않는 한 다른 리전에 존재하지 않습니다. 예를 들어, Amazon S3에서는 교차 리전 복제를 지원합니다. IAM과 같은 일부 서비스에는 교차 리전 리소스가 없습니다.	✓	✓	✓
해결 방법	단일 비디오 프레임의 픽셀 수를 설명합니다. 예를 들어 Full HD 또는 1080p는 1920x1080 픽셀로 프레임을 정의합니다.	✓	✓	
루트 사용자	AWS 계정 소유자입니다. 루트 사용자에게는 AWS 계정의 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있습니다.	✓	✓	✓
RTMP, RTMPS	실시간 메시징 프로토콜은 네트워크를 통한 오디오, 비디오 및 데이터 전송 관련 업계 표준입니다. RTMPS는 전송 계층 보안(TLS/SSL) 연결을 통해 실행되는 안전한 RTMP 버전입니다.	✓	✓	
S3 버킷	Amazon S3에 저장된 객체 모음입니다. 액세스 및 복제를 포함하여 많은 정책이 버킷 수준에서 정의되며 버킷의 모든 객체에 적용됩니다. 예를 들어 IVS 브로드캐스트는 S3 버킷에 여러 객체로 저장됩니다.	✓		
SDK	IVS로 애플리케이션을 구축하는 개발자를 위한 라이브러리 모음인 소프트웨어 개발 키트입니다.	✓	✓	✓

용어	설명	LL	RT	채팅
셀카 분할	카메라 이미지를 입력으로 허용하고 이미지의 각 픽셀에 대한 신뢰도 점수를 제공하는 마스크를 반환하여 해당 이미지가 전경 또는 배경에 있는지를 나타내는 클라이언트별 솔루션을 사용하여 라이브 스트림의 배경을 교체할 수 있습니다. IVS 브로드캐스트 SDK: 타사 카메라 필터(실시간 스트리밍)의 배경 교체를 참조하세요.		✓	
의미 체계 버전 관리	Major.Minor.Patch 형태의 버전 형식입니다. API에 영향을 주지 않는 버그 수정은 패치 버전을 증가시키고, 이전 버전과 호환되는 API 추가/변경은 마이너 버전을 증가시키며, 이전 버전과 호환되지 않는 API 변경은 메이저 버전을 증가시킵니다.	✓	✓	✓
서버 측 구성	IVS 서버를 사용하여 스테이지 참가자의 오디오와 비디오를 혼합한 다음에 이 혼합된 비디오를 IVS 채널로 보내 더 많은 대상에게 도달하거나 S3 버킷에 저장합니다. 서버 측 구성은 클라이언트 부하를 줄이고 브로드캐스트의 복원력을 개선하며 대역폭을 더 효율적으로 사용할 수 있도록 합니다. 클라이언트 측 구성도 참조하세요.		✓	
서비스 할당량	한 곳에서 많은 AWS 서비스에 대한 할당량을 관리하는 데 도움이 되는 AWS 서비스입니다. 할당량 값을 조회하는 것과 함께 Service Quotas 콘솔에서 할당량 증량을 요청할 수도 있습니다.	✓	✓	✓
서비스 연결 역할	AWS 서비스에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 IVS에서 자동으로 생성되며, 서비스에서 다른 AWS 서비스를 직접적으로 호출하는 데 필요한 모든 권한(예: S3 버킷 액세스 권한)을 포함합니다. IVS 보안의 IVS에 대해 서비스 연결 역할 사용을 참조하세요.	✓		

용어	설명	LL	RT	채팅
단계	실시간 이벤트 참가자가 실시간으로 비디오를 교환할 수 있는 가상 공간을 나타내는 IVS 리소스입니다. IVS 실시간 스트리밍 시작하기의 스테이지 생성을 참조하세요.		✓	
스테이지 세션	첫 번째 참가자가 스테이지에 조인하면 스테이지가 시작되고 마지막 참가자가 스테이지에 게시하는 것를 중지하면 몇 분 후에 종료됩니다. 수명이 긴 스테이지에는 수명 동안 여러 세션이 있을 수 있습니다.		✓	
스트림	소스에서 대상으로 지속적으로 보내지는 비디오 또는 오디오 콘텐츠를 나타내는 데이터입니다.	✓	✓	
스트림 키	채널을 생성할 때 IVS에서 할당하는 식별자입니다. 채널에 스트리밍 권한을 부여하는 데 사용됩니다. 스트림 키는 누구나 이 키를 통해 채널로 스트리밍할 수 있으므로 암호처럼 취급합니다. IVS 지연 시간이 짧은 스트리밍 시작하기를 참조하세요.	✓		
스트림 결핍	IVS로의 스트림 전송 지연 또는 중단입니다. 인코딩 디바이스에서 특정 기간에 전송할 것으로 광고한 예상 비트 양을 IVS에 수신되지 않으면 발생합니다. 스트림 결핍이 발생하면 스트림 결핍 이벤트가 됩니다. 시청자의 관점에서는 스트림 결핍이 지연, 버퍼링 또는 고정이 발생하는 비디오로 보일 수 있습니다. 스트림 결핍은 스트림 결핍을 초래한 특정 상황에 따라 짧을 수도 있고(5초 미만) 길 수도 있습니다(몇 분). 문제 해결 FAQ의 스트림 결핍이란 무엇인가요를 참조하세요.	✓	✓	
스트리머	IVS로 비디오 또는 오디오 스트림을 보내는 사람 또는 디바이스입니다.	✓	✓	

용어	설명	LL	RT	채팅
구독자	호스트의 비디오 및/또는 오디오를 수신하는 실시간 이벤트 참가자입니다. IVS 실시간 스트리밍이란을 참조하세요.		✓	
태그	AWS 리소스에 할당하는 메타데이터 레이블입니다. 태그를 사용하면 AWS 리소스를 식별하고 정리하는 데 도움이 됩니다. IVS 설명서 랜딩 페이지에서 실시간 스트리밍, 저지연 스트리밍 또는 채팅에 대한 IVS API 설명서의 '태그 지정'을 참조하세요.	✓	✓	✓
타사 카메라 필터	애플리케이션에서 이미지를 브로드캐스트 SDK에 사용자 지정 이미지 소스로 제공하기 전에 처리할 수 있도록 IVS 브로드캐스트 SDK와 통합할 수 있는 소프트웨어 구성 요소입니다. 타사 카메라 필터에서는 카메라의 이미지를 처리하고 필터 효과를 적용하는 등의 작업을 수행할 수 있습니다.	✓	✓	
썸네일	스트림에서 촬영한 축소된 크기의 이미지입니다. 기본적으로 60초마다 썸네일이 생성되지만, 더 짧은 간격을 구성할 수 있습니다. 썸네일 해상도는 채널 유형에 따라 다릅니다. Amazon S3에 자동 레코딩(저지연 스트리밍)의 레코딩 콘텐츠를 참조하세요.	✓		
시한 메타데이터	스트림 내 특정 타임스탬프에 연결된 메타데이터입니다. IVS API를 사용하여 프로그래밍 방식으로 추가하고 특정 프레임과 연결할 수 있습니다. 그러면 모든 시청자가 스트림을 기준으로 동일한 지점에서 메타데이터를 수신하게 됩니다. 시간 지정 메타데이터를 사용하여 스포츠 이벤트 중 팀 통계 업데이트와 같은 작업을 클라이언트에서 트리거할 수 있습니다. 비디오 스트림에 메타데이터를 포함하기를 참조하세요.	✓		

용어	설명	LL	RT	채팅
트랜스코딩	비디오와 오디오의 형식을 변환합니다. 수신 스트림은 다양한 재생 장치 및 네트워크 조건을 지원하기 위해 여러 비트 전송률과 해상도로 다른 형식으로 트랜스코딩될 수 있습니다.	✓	✓	
트랜스먹싱	비디오 스트림을 다시 인코딩하지 않는 IVS에 수집한 스트림의 간단한 리패키징입니다. '트랜스먹스'는 원래 스트림의 일부 또는 모두를 유지하면서 오디오 및/또는 비디오 파일의 형식을 변경하는 프로세스인 트랜스코딩 멀티플렉싱의 약자입니다. 트랜스먹스는 파일 내용을 변경하지 않고 다른 컨테이너 형식으로 변환됩니다. 트랜스코딩과 구별됩니다.	✓	✓	
변형 스트림	동일한 브로드캐스트를 여러 가지 품질 수준으로 인코딩한 세트입니다. 각 변형 스트림은 별도의 HLS 재생 목록으로 인코딩됩니다. 사용 가능한 변형 스트림의 인덱스를 다변량 재생 목록이라고 합니다. IVS 플레이어에서는 IVS로부터 다변량 재생 목록을 수신한 후 재생하는 동안 네트워크 상태 변화에 따라 앞뒤로 원활하게 변경되도록 변형 스트림 중에서 선택할 수 있습니다.	✓		
VBR	필요한 세부 수준에 따라 재생 내내 변경되는 동적 비트레이트를 사용하는 인코더의 속도 제어 방법인 가변 비트레이트입니다. 비디오 품질 문제 때문에 VBR은 사용하지 않는 것이 좋습니다. 그 대신에 CBR을 사용하세요.	✓	✓	

용어	설명	LL	RT	채팅
보기	비디오를 적극적으로 다운로드하거나 재생하는 고유한 시청 세션입니다. 조회수는 동시 보기 할당량의 기준입니다. 보기 세션이 비디오 재생을 시작하면 보기가 시작됩니다. 보기 세션이 비디오 재생을 중지하면 보기가 종료됩니다. 재생은 시청자의 유일한 지표입니다. 오디오 레벨, 브라우저 탭 포커스 및 비디오 품질과 같은 참여 경험적 방법은 고려되지 않습니다. 조회수를 계산할 때 IVS에서는 개별 시청자의 적합성을 고려하거나, 현지화된 시청자(예: 단일 머신의 여러 비디오 플레이어)를 중복 제거하려고 시도하지 않습니다. Service Quotas(저지연 스트리밍)의 기타 할당량을 참조하세요.	✓		
뷰어	IVS의 스트림을 수신하는 사람입니다.	✓		
WebRTC	웹 브라우저와 모바일 애플리케이션에 실시간 통신을 제공하는 오픈 소스 프로젝트인 웹 실시간 통신입니다. 플러그인을 설치하거나 네이티브 앱을 다운로드할 필요 없이 직접 통신이 가능하여 웹 페이지 내에서 오디오 및 비디오 peer-to-peer 통신이 작동할 수 있습니다. WebRTC의 기반이 되는 기술은 개방형 웹 표준으로 구현되며 모든 주요 브라우저에서 JavaScript 일반 API로 제공되거나 Android 및 iOS와 같은 네이티브 클라이언트를 위한 라이브러리로 제공됩니다.	✓	✓	

용어	설명	LL	RT	채팅
채찍	WebRTC-HTTP 통합 프로토콜은 WebRTC 기반의 콘텐츠 인제스트를 스트리밍 서비스 및/또는 CDN으로 허용하는 HTTP 기반 프로토콜입니다. WHIP는 WebRTC 사용을 표준화하기 위해 개발된 IETF 초안입니다. WHIP는 OBS와 같은 소프트웨어와의 호환성을 지원하여 데스크탑 퍼블리싱을 위한 대안 (IVS 브로드캐스트 SDK 대체) 을 제공합니다. OBS에 익숙한 좀 더 수준 높은 스트리머라면 장면 전환, 오디오 믹싱, 오버레이 그래픽과 같은 고급 제작 기능을 갖춘 OBS를 선호할 수도 있습니다. WHIP는 IVS 브로드캐스트 SDK를 사용하는 것이 불가능하거나 선호되지 않는 상황에서도 유용합니다. 예를 들어 하드웨어 인코더와 관련된 설정에서는 IVS 브로드캐스트 SDK가 옵션이 아닐 수 있습니다. 하지만 인코더가 WHIP를 지원하는 경우에도 인코더에서 IVS로 직접 게시할 수 있습니다. OBS 및 WHIP 지원을 참조하십시오.		✓	
WSS	WebSocket 보안은 암호화된 TLS WebSockets 연결을 통해 설정하기 위한 프로토콜입니다. IVS 챗 엔드포인트에 연결하는 데 사용됩니다. IVS 챗 시작하기의 Step 4: Send and Receive Your First Message를 참조하세요.			✓

문서 기록(실시간 스트리밍)

실시간 스트리밍 사용 설명서 변경 사항

변경 사항	설명	날짜
OBS 및 WHIP Support	새 페이지가 추가되었습니다. 이 문서에서는 OBS와 같은 Whip 호환 인코더를 사용하여 IVS 실시간 스트리밍에 게시하는 방법을 설명합니다. WHIP (WebRTC-HTTP 통합 프로토콜) 는 WebRTC 통합을 표준화하기 위해 개발된 IETF 초안입니다.	2024년 2월 6일
브로드캐스트 SDK: 안드로이드 1.14.1, iOS 1.14.1, 웹 1.8.0	real-time-streaming 브로드캐스트 SDK 가이드 (Android, iOS, 웹) 에서 새 릴리스의 버전 번호 및 아티팩트 링크가 업데이트되었습니다. Amazon IVS 설명서 랜딩 페이지에서 새 버전을 가리키는 브로드캐스트 SDK 참조 링크가 업데이트되었습니다. 이 릴리스에 대한 Amazon IVS 릴리스 정보도 참조하세요. Android 가이드에는 알려진 문제 (동영상 크기 176x176 미만) 를 새로 추가했습니다. 웹 가이드에는 알려진 문제가 새로 추가되었습니다. 해결 방법은 또는 를 호출할 때 비디오 해상도를 720p로 제한하는	2024년 2월 1일

	것입니다. `getUserMedia` `getDisplayMedia` 실시간 스트리밍 최적화에서 Simulcast를 사용한 레이어 인코딩 구성을 업데이트했습니다. 이제 이 기능은 기본적으로 비활성화되어 있습니다.	
브로드캐스트 SDK: 안드로이드 1.13.4, iOS 1.13.4, 웹 1.7.0	real-time-streaming 브로드캐스트 SDK 가이드 (Android, iOS, 웹) 에서 새 릴리스의 버전 번호 및 아티팩트 링크가 업데이트되었습니다. Amazon IVS 설명서 랜딩 페이지에서 새 버전을 가리키는 브로드캐스트 SDK 참조 링크가 업데이트되었습니다. 이 릴리스에 대한 Amazon IVS 릴리스 정보도 참조하세요.	2024년 1월 3일
IVS 용어집	IVS 실시간, 저지연 및 채팅 용어를 다루는 용어집을 확장했습니다.	2023년 12월 20일

스테이지 헬스: 새로운 CloudWatch 지표	PacketLoss (스테이지) 지표의 이름을 (Stage) 로 변경하고 IVS 실시간 스트리밍을 위한 추가 CloudWatch 지표를 공개했습니다. DownloadPacketLoss • DownloadPacketLoss (스테이지, 참가자) • DroppedFrames (스테이지, 참가자) • SubscribeBitrate (스테이지, 참가자,MediaType) IVS 실시간 스트리밍 모니터링을 참조하세요.	2023년 12월 7일
IAM 관리형 정책	두 개의 관리형 정책인 IVS ReadOnlyAccess 및 FullAccess IVS가 추가되었습니다. 다음을 참조하세요. • 보안 페이지에 있는 Amazon IVS에 대한 관리형 정책의 새 섹션입니다. • IVS 지연 시간이 짧은 스트리밍 시작하기의 3단계: IAM 권한 설정에 적용된 변경 사항입니다.	2023년 12월 5일

브로드캐스트 SDK: Android 1.13.2, iOS 1.13.2	real-time-streaming 브로드캐스트 SDK 가이드 Android 및 iOS에서 새 릴리스의 버전 번호 및 아티팩트 링크가 업데이트되었습니다. Amazon IVS 설명서 랜딩 페이지에서 새 버전을 가리키는 브로드캐스트 SDK 참조 링크가 업데이트되었습니다. 이 릴리스에 대한 Amazon IVS 릴리스 정보도 참조하세요.	2023년 12월 4일
브로드캐스트 SDK: Android 1.13.1	real-time-streaming 브로드캐스트 SDK 가이드: Android에서 새 릴리스의 버전 번호와 아티팩트 링크를 업데이트했습니다. Amazon IVS 설명서 랜딩 페이지에서 새 버전을 가리키는 브로드캐스트 SDK 참조 링크가 업데이트되었습니다. 이 릴리스에 대한 Amazon IVS 릴리스 정보도 참조하세요.	2023년 11월 21일
Service Quotas	"참가자 게시 해상도"를 1080p에서 720p로 변경했습니다.	2023년 11월 18일

브로드캐스트 SDK: Android 1.13.0, iOS 1.13.0	real-time-streaming 브로드캐스트 SDK 가이드 Android 및 iOS에서 새 릴리스의 버전 번호 및 아티팩트 링크가 업데이트되었습니다. Amazon IVS 설명서 랜딩 페이지에서 새 버전을 가리키는 브로드캐스트 SDK 참조 링크가 업데이트되었습니다. 이 릴리스에 대한 Amazon IVS 릴리스 정보도 참조하세요. 스트리밍 최적화에도 다양한 업데이트를 적용했습니다. 무엇보다도 "적응형 스트리밍: 동시 방송을 사용한 계층화된 인코딩" 기능은 이제 명시적 옵트인이 필요하며 최신 버전의 SDK에서만 지원됩니다.	2023년 11월 17일
복합 레코딩	다음과 같은 변경이 이루어졌습니다. • 이 새 기능에 복합 레코딩 페이지를 추가했습니다. • "IAM 권한 설정"의 정책에서 S3 엔드포인트를 사용한 IVS 실시간 스트리밍 시작하기를 업데이트했습니다. • Service Quotas에 새 엔드포인트에 대한 호출 비율 할당량을 업데이트했습니다.	2023년 11월 16일

서버 측 구성(SSC)	IVS 서버 측 구성은 클라이언트를 활성화하여 IVS 스테이지의 구성 및 브로드캐스팅을 IVS 관리형 서비스로 오프로드할 수 있습니다. 채널로 SSC 및 RTMP 브로드캐스트는 스테이지의 홈 리전에서 IVS 컨트롤 플레인 엔드포인트를 통해 호출됩니다. 다음을 참조하세요. • 시작하기 - "IAM 권한 설정"에서 정책에 SSC 엔드포인트를 추가했습니다. • EventBridgeAmazon과 IVS 사용 — 새 지표를 추가했습니다. • 서버 측 구성 - 이 새 문서는 개요 및 설정 지침을 포함하고 있습니다. • Service Quotas - 새로운 호출 비율 제한 및 기타 할당량을 추가했습니다. 다음 섹션도 참조하세요. • 변경 사항은 IVS 실시간 스트리밍 API 참조 변경 사항 아래에 나열되어 있습니다. • 변경 사항은 문서 기록(저지연 스트리밍)에 나열되어 있습니다.	2023년 11월 16일

IVS 브로드캐스트 SDK	브로드캐스트 SDK 개요에서 지원되는 SDK 버전을 명확히 하기 위해 플랫폼 요구 사항 > 네이티브 플랫폼을 업데이트하고 "모바일 브라우저(iOS 및 Android)"를 추가했습니다. 브로드캐스트 웹 가이드에 "모바일 웹 제한 사항"을 추가했습니다.	2023년 11월 9일
IVS 브로드캐스트 SDK	타사 카메라 필터에 새 페이지를 추가했습니다.	2023년 11월 9일
IVS 실시간 스트리밍 시작하기	IAM 권한 설정에서 절차를 업데이트했습니다.	2023년 10월 20일
실시간 스트리밍 모니터링	CloudWatch 지표: IVS 실시간 스트리밍에서 차원에 대한 샘플 값을 추가했습니다.	2023년 10월 17일
브로드캐스트 SDK: 웹 안내서	원격 참가자 미디어 음소거 상태 모니터링과 관련하여 변경된 내용이 몇 가지 있습니다.	2023년 10월 17일

브로드캐스트 SDK: 웹 1.6.0	real-time-streaming 브로드캐스트 SDK 가이드: Web에서 새 릴리스의 버전 번호와 아티팩트 링크를 업데이트했습니다. Amazon IVS 설명서 랜딩 페이지에서는 현재 버전의 브로드캐스트 SDK 참조를 가리킵니다. 이 릴리스에 대한 Amazon IVS 릴리스 정보도 참조하세요. 웹 가이드의 " MediaStream 디바이스에서 가져오기"에서도 두 `max` 줄을 삭제했습니다. 가장 좋은 방법은 지정만 하는 것입니다. `ideal` 실시간 스트리밍 최적화에서 오디오 비트레이트 및 스테레오 지원 최적화라는 섹션을 새로 추가했습니다.	2023년 10월 16일
스테이지 헬스: 새로운 CloudWatch 지표	IVS 실시간 스트리밍에 대한 CloudWatch 메트릭을 공개했습니다. IVS 실시간 스트리밍 모니터링을 참조하세요.	2023년 10월 12일

브로드캐스트 SDK: Android 1.12.1	real-time-streaming 브로드캐스트 SDK 가이드: Android에서 새 릴리스의 버전 번호 및 아티팩트 링크를 업데이트했습니다. 또한 Bluetooth 마이크 사용이라는 섹션을 새로 추가했습니다. Amazon IVS 설명서 랜딩 페이지에서는 현재 버전의 브로드캐스트 SDK 참조를 가리킵니다. 이 릴리스에 대한 Amazon IVS 릴리스 정보도 참조하세요.	2023년 10월 12일
브로드캐스트 SDK: 웹 1.5.2	real-time-streaming 브로드캐스트 SDK 가이드: Web에서 새 릴리스의 버전 번호와 아티팩트 링크를 업데이트했습니다. Amazon IVS 설명서 랜딩 페이지에서는 현재 버전의 브로드캐스트 SDK 참조를 가리킵니다. 이 릴리스에 대한 Amazon IVS 릴리스 정보도 참조하세요.	2023년 9월 14일
IVS 실시간 스트리밍 시작하기	Android > 브로드캐스트 SDK 설치에 데이터 바인딩을 추가했습니다.	2023년 9월 12일
브로드캐스트 SDK 오류 처리	브로드캐스트 SDK 가이드(웹, Android, iOS)에 “오류 처리” 섹션을 추가했습니다.	2023년 9월 12일

IVS 실시간 스트리밍 시작하기	참가자 토큰 배포에서 현재 토큰 형식을 기반으로 기능을 빌드하지 않는 것에 관한 중요 참고 사항이 추가되었습니다.	2023년 9월 1일
IVS 실시간 스트리밍 시작하기	IAM 권한 설정에서 권한 세트가 업데이트되었습니다.	2023년 8월 31일
브로드캐스트 SDK: 웹 1.5.1, Android 1.12.0 및 iOS 1.12.0	real-time-streaming 브로드캐스트 SDK 가이드 (웹, Android, iOS) 에서 새 릴리스의 버전 번호 및 아티팩트 링크가 업데이트되었습니다. Amazon IVS 설명서 랜딩 페이지에서 새 버전을 가리키는 브로드캐스트 SDK 참조 링크가 업데이트되었습니다. 이 릴리스에 대한 Amazon IVS 릴리스 정보도 참조하세요.	2023년 8월 23일
실시간 스트리밍 출시	이 릴리스에는 주요 설명서 변경 사항이 포함되어 있습니다. 이전 설명서의 이름을 IVS 지연 시간이 짧은 스트리밍으로 바꾸고 새 IVS 실시간 스트리밍 설명서를 게시했습니다. 이제 IVS 설명서 랜딩 페이지에 실시간 스트리밍과 지연 시간이 짧은 스트리밍에 대한 별도의 섹션이 있습니다. 각 섹션에는 자체 사용 설명서와 API 참조가 있습니다. 기타 설명서 변경 사항은 문서 기록(지연 시간이 짧은 스트리밍)을 참조하세요.	2023년 8월 7일

브로드캐스트 SDK: 웹 1.5.0, Android 1.11.0 및 iOS 1.11.0	브로드캐스트 SDK 가이드(웹, Android 및 iOS)에서 새 릴리스에 대한 버전 번호 및 아티팩트 링크가 업데이트되었습니다. Amazon IVS 설명서 랜딩 페이지에서 새 버전을 가리키는 브로드캐스트 SDK 참조 링크가 업데이트되었습니다. 이 릴리스에 대한 Amazon IVS 릴리스 정보도 참조하세요.	2023년 8월 7일

IVS Real-Time Streaming API Reference 변경 사항

API 변경 사항	설명	날짜
복합 레코딩	4개의 StorageConfiguration 엔드포인트와 7개의 객체 (DestinationDetail,, S3, S3Detail Recording Configuration, S3DestinationConfiguration,,) 를 추가했습니다. StorageConfiguration StorageConfiguration StorageConfigurationSummary 객체 3개 (컴포지션, 대상,) 를 수정했습니다. DestinationConfiguration 이는 응답과 StartComposition 요청 및 GetComposition 응답에 영향을 미칩니다.	2023년 11월 16일
서버 측 구성	8개의 컴포지션 및 EncoderConfiguration 엔드포인트와 11개의 개체 (ChannelDestinationConfiguration, 컴포지션, 대상,,, CompositionSummary,,,, DestinationConfiguration,, DestinationSummary,, EncoderConfiguration, EncoderConfigurationSummary,, GridConfiguration, LayoutConfiguration, 비디오) 를 추가했습니다.	2023년 11월 16일
스테이지 상태: 새로운 참가자 데이터	필드 6개(`browserName` , `browserVersion` , `ispName`, `osName`, `osVersion` 및 `sdkVersion`)	2023년 10월 12일

API 변경 사항	설명	날짜
	를 참가자 개체에 추가했습니다. 이는 GetParticipant 응답에 영향을 미칩니다.	
참가자 토큰	현재 토큰 형식을 기반으로 기능을 빌드하지 않는 것에 관한 중요 참고 사항이 추가되었습니다.	2023년 9월 1일
IVS 실시간 스트리밍 출시	이 릴리스에는 주요 설명서 변경 사항이 포함되어 있습니다. 이전 설명서의 이름을 IVS 지연 시간이 짧은 스트리밍으로 바꾸고 새 IVS 실시간 스트리밍 설명서를 게시했습니다. 이제 IVS 설명서 랜딩 페이지에 실시간 스트리밍과 지연 시간이 짧은 스트리밍에 대한 별도의 섹션이 있습니다. 각 섹션에는 자체 사용 설명서와 API 참조가 있습니다. IVS Real-Time Streaming API Reference는 IVS 실시간 스트리밍 설명서의 일부입니다. 이전에는 제목이 IVS Stage API Reference였습니다. 문서 기록(지연 시간이 짧은 스트리밍)에 이전 기록이 설명되어 있습니다.	2023년 8월 7일

릴리스 정보(실시간 스트리밍)

2024년 2월 6일

OBS 및 WHIP 지원

IVS는 OBS와 같은 WHIP 호환 인코더와 함께 사용하여 IVS 실시간 스트리밍에 게시할 수 있습니다. WHIP (WebRTC-HTTP 통합 프로토콜) 는 WebRTC 통합을 표준화하기 위해 개발된 IETF 초안입니다. OBS 및 WHIP 지원의 새 페이지를 참조하십시오.

2024년 2월 1일

아마존 IVS 브로드캐스트 SDK: 안드로이드 1.14.1, iOS 1.14.1, 웹 1.8.0 (실시간 스트리밍)

플랫폼	다운로드 및 변경 사항
웹 브로드캐스트 SDK 1.8.0	참조 문서: https://aws.github.io/ /docs/sdk-reference amazon-ivs-web-broadcast • 이제 멀티캐스트를 사용한 레이어 인코딩이 기본적으로 비활성화됩니다. • 스테이지가 삭제되거나 참가자가 서버와 연결이 끊겼을 때 Stage 인스턴스의 연결이 완전히 끊어지지 않는 문제를 수정했습니다. 이제 SDK에서 상태가 `DISCONNECTED` (및 이후 대신) 인 `STAGE_CONNECTION_STATE_CHANGED` 이벤트를 내보냅니다. `ERRORED CONNECTING` • 빈 오디오 또는 비디오 트랙으로 전략을 업데이트하면 게시가 실패하는 문제가 해결되었습니다.
안드로이드 브로드캐스트 SDK 1.14.1	참조 문서: https://aws.github.io/ /1.14.1/안드로이드 amazon-ivs-broadcast-docs

플랫폼	다운로드 및 변경 사항
	• 이제 동시 방송을 사용한 레이어 인코딩이 기본적으로 비활성화됩니다. • libWebRTC M108에서 M119로 업데이트되었습니다. • 몇 가지 충돌을 수정하여 전반적인 안정성을 개선했습니다. • 스테레오 퍼블리싱에 대한 지원이 추가되었습니다. StageAudioConfiguration 개체를 통해 활성화할 수 있습니다. • 세션에 참여한 후 참가자로부터 블랙 피드가 나오는 버그를 수정했습니다. • 동일한 호스트 애플리케이션에 다른 libWebRTC 버전이 포함되어 있을 때 기호 충돌이 발생하지 않도록 내부 libWebRTC 참조를 업데이트했습니다.

플랫폼	다운로드 및 변경 사항
iOS 브로드캐스트 SDK 1.14.1	실시간 스트리밍을 위해 다운로드: https://broadcast.live-video.net/1.14.1/AmazonIVSBroadcast-Stages.xcframework.zip 참조 문서: https://aws.github.io/ amazon-ivs-broadcast-docs /1.14.1/ios • 이제 동시 방송을 사용한 레이어 인코딩이 기본적으로 비활성화됩니다. • `libWebRTC` M108에서 M119로 업데이트되었습니다. • 몇 가지 충돌을 수정하여 전반적인 안정성을 개선했습니다. • 스테레오 퍼블리싱에 대한 지원이 추가되었습니다. 를 통해 활성화할 수 `IVSLocalStageStreamAudioConfiguration` 있습니다. • 다른 참가자를 위한 오디오 전용 모드를 활성화할 때 충돌이 발생하는 문제를 수정했습니다. • TTV를 개선하고 바이너리 크기를 줄였습니다.

브로드캐스트 SDK 크기: Android

아키텍처	압축된 크기	압축되지 않은 크기
arm64-v8a	5.23메가바이트	13.118 메가바이트
armeabi-v7a	4.524 메가바이트	9.134 메가바이트
x86_64	5.418 메가바이트	13.955 메가바이트
x86	5.61 메가바이트	14.369 메가바이트

브로드캐스트 SDK 크기: iOS

아키텍처	압축된 크기	압축되지 않은 크기
arm64	3.350 메가바이트	7.790 메가바이트

2024년 1월 3일

아마존 IVS 브로드캐스트 SDK: 안드로이드 1.13.4, iOS 1.13.4, 웹 1.7.0 (실시간 스트리밍)

플랫폼	다운로드 및 변경 사항
웹 브로드캐스트 SDK 1.7.0	참조 문서: https://aws.github.io/ /docs/sdk-reference amazon-ivs-web-broadcast • 스테이지에 참여하는 구독자를 위해 개선되었습니다 time-to-video . • `minAudioBitrateKbps` 속성을 삭제했습니다 (미사용). • 인터넷 중단 또는 변경 시 네트워크 복구가 개선되었습니다.
안드로이드 브로드캐스트 SDK 1.13.4	참조 문서: https://aws.github.io/ /1.13.4/안드로이드 amazon-ivs-broadcast-docs • StageAudioConfiguration 이제 에코 캔슬링 활성화 여부 설정을 지원합니다.
iOS 브로드캐스트 SDK 1.13.4	실시간 스트리밍을 위해 다운로드: https://broadcast.live-video.net/1.13.4/AmazonIVSBroadcast-Stages.xcframework.zip 참조 문서: https://aws.github.io/ amazon-ivs-broadcast-docs /1.13.4/ios

플랫폼	다운로드 및 변경 사항
	• iOS에서는 안정성과 복구성에 중점을 두고 녹음과 재생을 위한 오디오 엔진을 개선했습니다. 이를 통해 사용 중 경로 변경에 대한 지원이 향상되고, 엣지 케이스의 배터리 복구 성능이 향상되며, 메인 스레드 차단량이 줄어듭니다. • 마이크를 스테이지에서 분리한 후에도 iOS 개인 정보 표시기가 켜진 상태로 유지되는 문제를 수정했습니다. (당시 SDK는 수신 오디오를 처리하지 않았습니다.)

브로드캐스트 SDK 크기: Android

아키텍처	압축된 크기	압축되지 않은 크기
arm64-v8a	5.187메가바이트	13.025 메가바이트
armeabi-v7a	4.491 메가바이트	9.056 메가바이트
x86_64	5.359 메가바이트	13.829 메가바이트
x86	5.553 메가바이트	14.214 메가바이트

브로드캐스트 SDK 크기: iOS

아키텍처	압축된 크기	압축되지 않은 크기
arm64	3.45MB	7.84MB

2023년 12월 7일

새 메트릭스 CloudWatch

PacketLoss (스테이지) 지표의 이름을 (스테이지) 로 DownloadPacketLoss 변경했습니다. 또한 IVS 실시간 스트리밍에 대한 추가 CloudWatch 지표도 공개했습니다.

- DownloadPacketLoss (스테이지, 참가자)
- DroppedFrames (스테이지, 참가자)
- SubscribeBitrate (무대, 참가자,MediaType)

자세한 내용은 Amazon IVS 실시간 스트리밍 모니터링을 참조하세요.

2023년 12월 4일

Amazon IVS 브로드캐스트 SDK: Android 1.13.2 및 iOS 1.13.2(실시간 스트리밍)

플랫폼	다운로드 및 변경 사항
모든 모바일(Android 및 iOS)	• 개발자는 게시 활성화/비활성화에 노이즈 억제 구성을 사용할 수 있습니다.
Android 브로드캐스트 SDK 1.13.2	참조 문서: https://aws.github.io/ amazon-ivs-broadcast-docs /1.13.2/안드로이드 • 세션의 첫 번째 스테이지에 조인할 때 비디오(TTV)를 로드하는 데 걸리는 시간을 개선했습니다.
iOS 브로드캐스트 SDK 1.13.2	실시간 스트리밍을 위한 다운로드: https://broadcast.live-video.net/1.13.2/AmazonIVSBroadcast-Stages.xcframework.zip 참조 문서: https://aws.github.io/ amazon-ivs-broadcast-docs /1.13.2/ios

플랫폼	다운로드 및 변경 사항
	• 실시간 SDK에는 변경 사항이 없습니다.

브로드캐스트 SDK 크기: Android

아키텍처	압축된 크기	압축되지 않은 크기
arm64-v8a	5.177MB	13.01MB
armeabi-v7a	4.485MB	9.045MB
x86_64	5.352MB	13.808MB
x86	5.547MB	14.192MB

브로드캐스트 SDK 크기: iOS

아키텍처	압축된 크기	압축되지 않은 크기
arm64	3.45MB	7.82MB

2023년 11월 21일

Amazon IVS 브로드캐스트 SDK: Android 1.13.1(실시간 스트리밍)

플랫폼	다운로드 및 변경 사항
Android 브로드캐스트 SDK 1.13.1	참조 문서: https://aws.github.io/ /1.13.1/안드로이드 amazon-ivs-broadcast-docs • 같은 스테이지에서 빠르게 떠날 때, 릴리스할 때, 다시 참여할 때 충돌이 발생하는 문제를 수정했습니다.

브로드캐스트 SDK 크기: Android

아키텍처	압축된 크기	압축되지 않은 크기
arm64-v8a	5.177MB	13.102MB
armeabi-v7a	4.485MB	9.046MB
x86_64	5.353MB	13.809MB
x86	5.547MB	14.192MB

2023년 11월 17일

Amazon IVS 브로드캐스트 SDK: Android 1.13.0 및 iOS 1.13.0(실시간 스트리밍)

플랫폼	다운로드 및 변경 사항
모든 모바일(Android 및 iOS)	• 업데이트된 스트리밍 최적화 무엇보다도 "적응형 스트리밍: 동시 방송을 사용한 계층화된 인코딩" 기능은 이제 명시적 옵트인이 필요하며 최신 버전의 SDK에서만 지원됩니다. • 드물게 발생하는 충돌을 줄여 스테이지의 안정성이 개선되었습니다. • 스테이지에 조인할 때 비디오(TTV)를 로드하는 데 걸리는 시간을 개선했습니다. • Bluetooth 디바이스 사용 경험을 개선했습니다. • SDK CPU 및 메모리 사용을 최적화하고 라이브러리 크기를 줄였습니다. • 음성 통신, 미디어 재생 등에 대한 사전 설정을 포함하여 오디오 캡처 및 재생 파라미터를 설정하는 데 사용할 수 있는 `StageAudioManager` 클래스가 추가되었습니다. 자

플랫폼	다운로드 및 변경 사항
	세한 내용은 새 페이지인 IVS 브로드캐스트 SDK: 모바일 오디오 모드를 참조하세요. • WebRTC 통계에서 구조화된 품질 이벤트를 표시하는 새로운 `requestQualityStats` 기능을 추가했습니다. • 오디오 비트레이트를 업데이트하는 새 함수를 추가했습니다. 비디오 구성과 마찬가지로 새 오디오 구성 개체를 통해 `LocalStageStream` 개체에 설정됩니다.

플랫폼	다운로드 및 변경 사항

Android 브로드캐스트 SDK 1.13.0	참조 문서: https://aws.github.io/ /1.13.0/안드로이드 amazon-ivs-broadcast-docs
	• `StageRenderer` 인터페이스의 모든 메서드는 이제 선택 사항입니다.
	• 향상된 성능을 위해 `Surfaceview` 기반 미리 보기에 대한 지원이 추가되었습니다. `Session` 및 `StageStream` 의 기존 `getPreview` 메서드는 계속 `TextureView` 의 서브클래스를 반환하지만, 이는 향후 SDK 버전에서 변경될 수 있습니다.
	  • 애플리케이션이 특정 `TextureView` 에 의존하는 경우 변경 없이 계속할 수 있습니다. `getPreview` 에서 `getPreviewTextureView` 로 전환하여 기본값 `getPreview` 가 반환되는 최종 변경에 대비할 수도 있습니다.
	  • 응용 프로그램에서 특별히 `TextureView` 를 요구하지 않는 경우 CPU 및 메모리 사용량을 낮추려면 `getPreviewSurfaceView` 로 전환하는 것이 좋습니다.
	• 이제 SDK는 애플리케이션에서 제공하는 Android Surface 객체와 작동하며 `ImagePreviewSurfaceTarget` 이라고 하는 새로운 유형의 미리보기를 구현합니다. 더 나은 유연성을 제공하는 Android View의 하위 클래스가 아닙니다.
	• 원격 참가자에 대한 `onFrame` 콜백이 잘못된 시간에 잘못된 크기로 호출되는 사례를 수정했습니다.
	• 이제 `SurfaceSource # getInputSurface` 에 `@Nullable` 주석이 추가되었습

플랫폼	다운로드 및 변경 사항
	니다. 코드를 사용하기 전에 코드를 확인해야 합니다. • UserId 및 attributes 가 ParticipantInfo 에 추가되었습니다. UserId 및 attributes 속성은 토큰에 임베디드되어 있으며 참가자가 참여할 때마다 ParticipantInfo 를 통해 애플리케이션이 이를 검색할 수 있습니다. • 이제 카메라 캡처 및 미리 보기 렌더링은 기본적으로 720 x 1280 또는 게시 해상도(둘 중 더 큰 값)가 15fps로 기본 설정됩니다. StageVideoConfiguration # setCameraCaptureQuality 을 사용하여 해상도와 fps를 조정할 수 있습니다. • 구성 속성을 설정할 때 발생하는 IllegalArgumentException 에 이제 예외 메시지에 제공된 값이 포함됩니다.

플랫폼	다운로드 및 변경 사항
iOS 브로드캐스트 SDK 1.13.0	실시간 스트리밍을 위한 다운로드: https://broadcast.live-video.net/1.13.0/AmazonIVSBroadcast-Stages.xcframework.zip 참조 문서: https://aws.github.io/ amazon-ivs-broadcast-docs /1.13.0/ios • 게시하기 전에 비디오 구성을 업데이트하면 SDK가 비디오 구성을 변경하지 않는 문제를 수정했습니다. • LibVPX 보안 취약성(CVE-2023-5217)에 대한 Google 수정 사항을 통합했습니다. (참고로 Android SDK는 이 문제와 관련하여 어떠한 변경도 요구하지 않았습니다.) • `libWebRTC` 를 포함하는 다른 라이브러리를 사용하는 애플리케이션은 더 이상 IVS 브로드캐스트 SDK와 충돌하지 않습니다. • 이제 `IVSStageRenderer` 프로토콜의 모든 메서드가 `@optional` 으로 표시됩니다. • 이제 SDK에서 반환한 마이크와 카메라의 정렬 순서가 보장됩니다. 이는 SDK 자체에 설명되어 있습니다. • 이제 여러 카메라가 운영 체제에 따라 위치별로 하나씩 `isDefault` 속성의 `true` 값을 가질 수 있습니다. • 기본 `AVAudioSession` 을 정밀하게 제어하여 스테이지 기능을 더 다양하게 사용할 수 있도록 하는 `IVSStageAudioManager` 가 추가되었습니다. • `ParticipantInfo` 에 `UserId`가 추가되었습니다.

브로드캐스트 SDK 크기: Android

아키텍처	압축된 크기	압축되지 않은 크기
arm64-v8a	5.17MB	13.00MB
armeabi-v7a	4.48MB	9.04MB
x86_64	5.35MB	13.80MB
x86	5.54MB	14.18MB

브로드캐스트 SDK 크기: iOS

아키텍처	압축된 크기	압축되지 않은 크기
arm64	3.45MB	7.84MB

2023년 11월 16일

복합 레코딩

이 새로운 기능을 사용하면 Amazon S3 버킷에 IVS Stage의 복합 보기를 레코딩할 수 있습니다. 자세한 내용은 다음을 참조하십시오.

- 복합 레코딩 - 새 페이지입니다.
- IVS 실시간 스트리밍 시작하기 - "IAM 권한 설정"에 있는 정책에 S3 엔드포인트를 추가했습니다.
- Service Quotas - 새 엔드포인트에 대한 호출 비율 할당량을 추가했습니다.
- IVS 실시간 스트리밍 API 레퍼런스 — 4개의 StorageConfiguration 엔드포인트와 7개의 객체 (DestinationDetail,, S3, S3Detail, S3DestinationConfiguration, RecordingConfiguration,) 를 추가했습니다. StorageConfiguration StorageConfiguration StorageConfigurationSummary 또한 3개의 객체 (컴포지션, 대상, DestinationConfiguration) 를 수정했는데, 이는 응답과 요청 및 GetComposition 응답에 영향을 줍니다. StartComposition

2023년 11월 16일

서버 측 구성

IVS 서버 측 구성은 클라이언트를 활성화하여 IVS 스테이지의 구성 및 브로드캐스팅을 IVS 관리형 서비스로 오프로드할 수 있습니다. 서버 측 구성 및 채널로 RTMP 브로드캐스트는 스테이지의 홈 리전에서 IVS 제어 플레인 엔드포인트를 통해 호출됩니다. 자세한 내용은 다음을 참조하십시오.

- IVS 실시간 스트리밍 시작하기 - "IAM 권한 설정"에서 정책에 SSC 엔드포인트를 추가했습니다.
- EventBridge Amazon과 IVS 실시간 스트리밍 사용 — 새 지표를 추가했습니다.
- 서버 측 구성 - 이 새 문서는 개요 및 설정 지침을 포함하고 있습니다.
- Service Quotas(실시간 스트리밍) - 새로운 호출 비율 한도 및 기타 할당량을 추가했습니다.
- 실시간 스트리밍 API 레퍼런스 — 8개의 컴포지션 및 EncoderConfiguration 엔드포인트와 11개의 객체 (ChannelDestinationConfiguration, 컴포지션,, 대상, CompositionSummary,,,,, DestinationConfiguration, DestinationSummary, EncoderConfiguration,, EncoderConfigurationSummary, GridConfiguration LayoutConfiguration, 비디오) 를 추가했습니다.

IVS 저지연 스트리밍 사용 설명서에서 다음을 참조하세요.

- IVS 스트림에서 여러 호스트 활성화 - "스테이지 브로드캐스팅: 클라이언트 측 대 서버 측 구성"을 추가하고 "4. 스테이지 브로드캐스트"를 업데이트했습니다.

2023년 10월 16일

Amazon IVS 브로드캐스트 SDK: 웹 1.6.0(실시간 스트리밍)

플랫폼	다운로드 및 변경 사항
웹 브로드캐스트 SDK 1.6.0	참조 문서: https://aws.github.io/ amazon-ivs-web-broadcast /docs/sdk-reference • TTV(Time-To-Video)를 개선했습니다. • `maxAudioBitrate` 구성을 추가하여 최대 128kbps의 모노 또는 스테레오 오디오 채널을 지원합니다.

2023년 10월 12일

새 지표 및 참가자 데이터 CloudWatch

IVS 실시간 스트리밍에 대한 CloudWatch 지표를 공개했습니다. 자세한 내용은 Amazon IVS 실시간 스트리밍 모니터링을 참조하세요.

또한 참가자 API 개체에 필드 6개(`browserName`, `browserVersion`, `ispName`, `osName`, `osVersion` 및 `sdkVersion`)를 추가했습니다. 이는 GetParticipant 응답에 영향을 미칩니다. IVS 실시간 스트리밍 API 참조를 참조하세요.

2023년 10월 12일

Amazon IVS 브로드캐스트 SDK: Android 1.12.1(실시간 스트리밍)

플랫폼	다운로드 및 변경 사항
Android 브로드캐스트 SDK 1.12.1	참조 문서: https://aws.github.io/ amazon-ivs-broadcast-docs /1.12.1/안드로이드 • `BroadcastSession.setListener` 를 직접적으로 호출할 경우 오류가 발생하는 버그를 수정했습니다.

브로드캐스트 SDK 크기: Android

아키텍처	압축된 크기	압축되지 않은 크기
arm64-v8a	5.853MB	16.375MB
armeabi-v7a	4.895MB	10.803MB
x86_64	6.149MB	17.318MB
x86	6.328MB	17.186MB

2023년 9월 14일

Amazon IVS 브로드캐스트 SDK: 웹 1.5.2(실시간 스트리밍)

플랫폼	다운로드 및 변경 사항
웹 브로드캐스트 SDK 1.5.2	참조 문서: https://aws.github.io/ /docs/sdk-reference amazon-ivs-web-broadcast • 게시된 상태가 `ERRORED` 상태로 전환될 때 `refreshStrategy` 를 사용하여 다시 게시하지 못하게 하는 버그를 수정했습니다.

2023년 8월 23일

Amazon IVS 브로드캐스트 SDK: 웹 1.5.1, Android 1.12.0 및 iOS 1.12.0(실시간 스트리밍)

플랫폼	다운로드 및 변경 사항
웹 브로드캐스트 SDK 1.5.1	참조 문서: https://aws.github.io/ /docs/sdk-reference amazon-ivs-web-broadcast • TypeScript 5의 내부 Maybe 유형 관련 버그가 수정되었습니다. • Simulcast 지원에 대한 감지 기능이 향상되었습니다. • 게시하려고 할 때 `refreshStrategy` 관련 두 가지 경쟁 조건을 수정했습니다. • 구구독할 참가자를 업데이트하려고 할 때 `refreshStrategy` 의 경쟁 조건을 수정했습니다.
모든 모바일(Android 및 iOS)	• 간혹 게시 작업이 완료되지 않는 문제를 수정했습니다.

플랫폼	다운로드 및 변경 사항
	• 드물게 발생하는 충돌을 줄여 스테이지의 안정성을 개선했습니다. • 빠른 참가/나가기로 인해 발생하는 경쟁 조건 문제를 해결하여 스테이지의 안정성을 개선했습니다. • `ImageDevice` 에 새로운 `setOnFrameCallback` 메서드를 추가했습니다. 이를 통해 프레임이 디바이스 자체를 통과할 때 관찰할 수 있어 최신 이미지의 화면 비율을 파악할 수 있습니다. 이 메서드는 스테이지의 원격 참가자에 대한 첫 번째 프레임이 렌더링되는 시점을 감지하는 데에도 사용할 수 있습니다.
Android 브로드캐스트 SDK 1.12.0	참조 문서: https://aws.github.io/ amazon-ivs-broadcast-docs /1.12.0/안드로이드 • 이제 Android 9가 지원됩니다. • CPU 사용량 및 성능이 개선되었습니다.
iOS 브로드캐스트 SDK 1.12.0	실시간 스트리밍을 위한 다운로드: https://broadcast.live-video.net/1.12.0/AmazonIVSBroadcast-Stages.xcframework.zip 참조 문서: https://aws.github.io/ amazon-ivs-broadcast-docs /1.12.0/ios • `IVSCustomImageSource` 대신 `IVSCustomAudioSource` 를 반환하도록 `IVSDeviceDiscovery.createAudioSourceWithName` 의 서명을 수정했습니다.

브로드캐스트 SDK 크기: Android

아키텍처	압축된 크기	압축되지 않은 크기
arm64-v8a	5.853MB	16.375MB
armeabi-v7a	4.895MB	10.803MB
x86_64	6.149MB	17.318MB
x86	6.328MB	17.186MB

브로드캐스트 SDK 크기: iOS

아키텍처	압축된 크기	압축되지 않은 크기
arm64	5.06MB	10.92MB

2023년 8월 7일

Amazon IVS 브로드캐스트 SDK: 웹 1.5.0, Android 1.11.0 및 iOS 1.11.0

플랫폼	다운로드 및 변경 사항
웹 브로드캐스트 SDK 1.5.0	참조 문서: https://aws.github.io/ /docs/sdk-reference amazon-ivs-web-broadcast • 동시 방송 추가 - 이 기능을 활성화하면 게시자가 고품질 및 저품질 비디오 계층을 전송할 수 있습니다. 구독자는 네트워크 상태에 따라 최적의 품질을 자동으로 선택합니다. 미디어 최적화를 참조하세요.
모든 모바일(Android 및 iOS)	동시 방송 추가 - 이 기능을 활성화하면 게시자가 고품질 및 저품질 비디오 계층을 전송할 수 있습니다. 구독자는 네트워크 상태에 따라 최적

플랫폼	다운로드 및 변경 사항
	의 품질을 자동으로 선택합니다. Android 및 iOS 브로드캐스트 SDK 가이드의 '동시 방송을 사용한 계층화된 인코딩 활성화/비활성화'를 참조하세요.
Android 브로드캐스트 SDK 1.11.0	참조 문서: https://aws.github.io/ /1.11.0/안드로이드 amazon-ivs-broadcast-docs • 많은 스테이지를 생성하면 결국 충돌이 발생하는 문제를 수정했습니다. (정확한 스테이지 수는 디바이스에 따라 다릅니다.)
iOS 브로드캐스트 SDK 1.11.0	실시간 스트리밍용 다운로드: https://broadcast.live-video.net/1.11.0/AmazonIVSBroadcast-Stages.xcframework.zip 참조 문서: https://aws.github.io/ /1.11.0/ios amazon-ivs-broadcast-docs • `IVSCustomImageSource` 대신 `IVSCustomAudioSource` 를 반환하도록 `IVSDeviceDiscovery.createAudioSourceWithName` 의 서명을 수정했습니다.

브로드캐스트 SDK 크기: Android

아키텍처	압축된 크기	압축되지 않은 크기
arm64-v8a	5.811MB	16.186MB
armeabi-v7a	4.857MB	10.646MB
x86_64	6.108MB	17.122MB
x86	6.289MB	16.994MB

브로드캐스트 SDK 크기: iOS

아키텍처	압축된 크기	압축되지 않은 크기
arm64	5.030MB	10.810MB

2023년 8월 7일

실시간 스트리밍

Amazon Interactive Video Service(IVS) 실시간 스트리밍을 사용하면 300밀리초 미만의 지연 시간으로 호스트에게서 시청자에게 라이브 스트림을 전송할 수 있습니다.

이 릴리스에는 주요 설명서 변경 사항이 포함되어 있습니다. 이제 IVS 설명서 랜딩 페이지에 실시간 스트리밍과 지연 시간이 짧은 스트리밍에 대한 별도의 섹션이 있습니다. 각 섹션에는 자체 사용 설명서와 API 참조가 있습니다. 설명서에 대한 자세한 내용은 문서 기록(실시간 및 지연 시간이 짧은 스트리밍 설명서 모두)을 참조하세요. 실시간 스트리밍의 경우 IVS 실시간 스트리밍 사용 설명서와 IVS Real-Time Streaming API Reference로 시작하세요.

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.