aws

개발자 가이드

# AWS Device Farm

API 버전 2015-06-23

# AWS Device Farm: 개발자 가이드

# Table of Contents

# AWS Device Farm이란 무엇인가요?

Device Farm은 Amazon Web Services(AWS)에서 호스팅하는 Android, iOS 및 웹 애플리케이션을 실제 휴대폰 및 태블릿에서 테스트하고 상호 작용할 수 있도록 하는 앱 테스트 서비스입니다.

Device Farm을 사용하는 두 가지 주요 방법은 다음과 같습니다.

- 다양한 테스트 프레임워크를 사용하여 앱을 자동으로 테스트합니다.
- 실시간으로 앱을 로드하고, 실행하고, 상호 작용할 수 있는 디바이스에 원격으로 액세스할 수 있습니다.

> **Note**
>
> Device Farm은 `us-west-2`(오레곤) 리전에서만 사용할 수 있습니다.

## 자동 앱 테스트

Device Farm에서는 자체 테스트를 업로드하거나 스크립트가 필요 없는 내장된 호환성 테스트를 사용할 수 있습니다. 테스트는 병렬로 수행되기 때문에 여러 디바이스의 테스트가 몇 분 안에 시작됩니다.

테스트가 완료되면 상위 수준 결과, 하위 수준 로그, 픽셀 단위 스크린샷 및 성능 데이터가 포함된 테스트 보고서가 업데이트됩니다.

Device Farm은 PhoneGap, Titanium, Xamarin, Unity 및 기타 프레임워크로 만든 앱을 포함하여 네이티브 및 하이브리드 Android와 iOS 앱의 테스트를 지원합니다. 대화형 테스트를 위한 Android 및 iOS 앱의 원격 액세스를 지원합니다. 지원되는 DNS 유형에 대한 자세한 내용은 AWS Device Farm에서 테스트 유형을 사용한 작업 단원을 참조하세요.

## 원격 액세스 상호 작용

원격 액세스 기능을 사용하면 기능을 테스트하고 고객 문제를 재현하기 위해 웹 브라우저를 통해 실시간으로 살짝 밀기와 제스처를 수행하고 디바이스와 상호 작용할 수 있습니다. 디바이스와의 실시간 상호 작용이 유용한 상황이 많이 있습니다. 예를 들어 고객 서비스 담당자는 고객에게 디바이스 사용 또는 설정 과정을 안내할 수 있습니다. 또한 고객에게 특정 디바이스에서 실행되는 앱을 사용하는 방법을 안내할 수 있습니다. 원격 액세스 세션에서 실행 중인 디바이스에 앱을 설치한 다음 고객 문제 또는 보고된 버그를 재현할 수 있습니다.

원격 액세스 세션 중에 Device Farm은 사용자가 디바이스와 상호 작용할 때 발생하는 작업에 대한 세부 정보를 수집합니다. 세션이 끝날 때 이러한 세부 정보가 포함된 로그와 세션의 비디오 캡처가 생성됩니다.

# 용어

Device Farm에는 정보가 구성되는 방식을 정의하는 다음 용어가 도입되었습니다.

디바이스 풀

플랫폼, 제조업체 또는 모델 등 일반적으로 유사한 특성을 공유하는 디바이스 모음입니다.

작업

단일 디바이스에서 단일 앱을 테스트하기 위한 Device Farm에 대한 요청입니다. 작업에는 하나 이상의 제품군이 포함되어 있습니다.

측정

디바이스 요금 청구를 말합니다. 설명서 및 API 참조에서 미터링된 디바이스 또는 측정되지 않은 디바이스에 대한 참조를 볼 수 있습니다. 요금에 대한 자세한 내용은 AWS Device Farm 요금을 참조하세요.

프로젝트

실행을 포함하는 논리적 워크스페이스을 나타내며, 하나 이상의 디바이스에서 단일 앱의 각 테스트당 한 번씩 실행됩니다. 프로젝트를 사용하면 원하는 방식으로 워크스페이스을 조직할 수 있습니다. 예를 들어 앱 제목당 하나의 프로젝트 또는 플랫폼당 하나의 프로젝트가 있을 수 있습니다. 필요한 만큼 프로젝트를 생성할 수 있습니다.

보고서

하나 이상의 디바이스에 대해 단일 앱을 테스트하기 위한 Device Farm에 대한 요청인 실행에 대한 정보를 포함합니다. 자세한 내용은 AWS Device Farm에 있는 보고서 단원을 참조하세요.

실행

특정 디바이스 세트에서 실행될 특정 테스트 세트가 있는 앱의 특정 빌드입니다. 실행하면 결과 보고서가 생성됩니다. 실행에는 하나 이상의 작업이 포함되어 있습니다. 자세한 내용은 실행 단원을 참조하세요.

세션

웹 브라우저를 통해 실제 물리적 디바이스와 실시간으로 상호 작용합니다. 자세한 내용은 세션 단원을 참조하세요.

스위트

테스트 패키지 내 테스트의 계층적 구성 스위트에는 하나 이상의 테스트가 포함되어 있습니다.

테스트

테스트 패키지의 개별 테스트 케이스

Device Farm에 관한 자세한 내용은 개념 단원을 참조하세요.

# 설정

Device Farm을 사용하려면 설정을 참조하세요.

# AWS Device Farm 설정

Device Farm을 처음 사용하기 전에 다음의 작업을 완료해야 합니다.

주제



## 1단계: 가입 AWS

Amazon Web Services(AWS) 가입.

계정이 없는 경우 다음 단계를 완료하여 계정을 만드세요. AWS 계정

가입하려면 AWS 계정

1. https://portal.aws.amazon.com/billing/signup을 엽니다.
2. 온라인 지시 사항을 따릅니다.

   등록 절차 중 전화를 받고 전화 키패드로 확인 코드를 입력하는 과정이 있습니다.

   에 AWS 계정가입하면 AWS 계정 루트 사용자a가 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

## 2단계: AWS 계정에서 IAM 사용자 생성 또는 사용

AWS 루트 계정을 사용하여 Device Farm에 액세스하지 않는 것이 좋습니다. 대신 AWS 계정에서 AWS Identity and Access Management (IAM) 사용자를 생성 (또는 기존 사용자 사용) 한 다음 해당 IAM 사용자로 Device Farm에 액세스하십시오.

자세한 내용은 IAM 사용자 생성(AWS Management Console)을 참조하세요.

# 3단계: IAM 사용자에게 Device Farm 액세스 권한 부여

IAM 사용자에게 Device Farm에 액세스할 수 있는 권한 부여 이를 위해 다음과 같이 IAM에 액세스 정책을 만든 후 다음과 같이 액세스 정책을 IAM 사용자에게 할당하세요.

> **Note**
>
> 다음 단계를 완료하는 데 사용하는 AWS 루트 계정 또는 IAM 사용자는 다음 IAM 정책을 생성하여 IAM 사용자에게 연결할 수 있는 권한이 있어야 합니다. 자세한 내용은 정책 작업을 참조하세요.

1. 다음 JSON 본문을 사용하여 정책을 만드세요. 설명적인 제목 (예:) 을 지정하십시오. *DeviceFarmAdmin*

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "devicefarm:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

   IAM 정책 생성에 대한 자세한 내용은 IAM 사용 설명서의 IAM 정책 생성을 참조하세요.

2. 생성한 IAM 정책을 새 사용자에 연결하세요. 사용자 또는 그룹에 IAM 정책을 연결하는 방법에 대한 자세한 내용은 IAM 설명서의 IAM 정책 추가 및 제거를 참조하세요.

정책을 연결하면 IAM 사용자가 해당 사용자와 연결된 모든 Device Farm 작업 및 리소스에 액세스할 수 있습니다. IAM 사용자가 한정된 범위의 Device Farm 작업 및 리소스만 사용할 수 있도록 제한하는 방법은 AWS Device Farm의 Identity and Access Management 단원을 참조하세요.

# 다음 단계

이제 Device Farm을 사용할 준비가 되었습니다. Device Farm 시작하기 단원을 참조하세요.

# Device Farm 시작하기

Device Farm을 사용하여 기본 Android 또는 iOS 앱을 테스트하는 방법을 자세히 알아보세요. Device Farm 콘솔을 사용하여 프로젝트를 만들고, .apk 또는 .ipa 파일을 업로드하고, 표준 테스트 스위트를 실행한 다음 결과를 확인합니다.

> **Note**
>
> Device Farm은(`us-west-2`오레곤) AWS 리전에서만 사용할 수 있습니다.

주제



## 사전 조건

시작하기 전에 먼저 다음 요구 사항을 완료해야 합니다.

- 설정의 단계를 수행합니다. AWS 계정과 Device Farm에 액세스할 권한이 있는 AWS Identity and Access Management(IAM) 사용자가 필요합니다.

- Android의 경우 .apk(Android 앱 패키지) 파일이 필요합니다. iOS의 경우 .ipa(iOS 앱 아카이브) 파일이 필요합니다. 이 연습의 뒷부분에서 파일을 Device Farm에 업로드하세요.

  > **Note**
  >
  > .ipa 파일은 시뮬레이터가 아닌 iOS 디바이스용으로 빌드되어야 합니다.

- (선택 사항) Device Farm이 지원하는 테스트 프레임워크 중 하나의 테스트가 필요합니다. 이 테스트 패키지를 Device Farm에 업로드한 다음, 이 연습의 뒷부분에 테스트를 실행합니다. 사용할 테스트

패키지가 없는 경우 표준 기본 제공 테스트 제품군을 지정하고 실행할 수 있습니다. 자세한 내용은 AWS Device Farm에서 테스트 유형을 사용한 작업 단원을 참조하세요.

## 1단계: 콘솔에 로그인

Device Farm 콘솔을 사용하여 테스트용 프로젝트와 실행을 만들고 관리할 수 있습니다. 이 연습의 뒷부분에 프로젝트와 실행을 학습합니다.

- https://console.aws.amazon.com/devicefarm에서 Device Farm 콘솔에 로그인하세요.

## 2단계: 프로젝트 생성

Device Farm에서 앱을 테스트하려면 먼저 프로젝트를 만들어야 합니다.

1. 탐색 창에서 모바일 디바이스 테스트를 선택하고 프로젝트를 선택하세요.
2. 모바일 디바이스 테스트 프로젝트에서 새 프로젝트를 선택하세요.
3. 프로젝트 만들기에서 프로젝트 이름(예: **MyDemoProject**)을 입력하세요.
4. 생성을 선택하세요.

   콘솔은 새로 만든 프로젝트의 자동 테스트 페이지를 엽니다.

## 3단계: 실행 만들고 시작하기

이제 프로젝트가 있으므로 실행을 만들어 시작할 수 있습니다. 자세한 내용은 실행 단원을 참조하세요.

1. 자동 테스트 페이지에서 새 실행 생성을 선택하세요.
2. 애플리케이션 선택 페이지의 모바일 앱에서 파일 선택을 선택한 다음 컴퓨터에서 Android(.apk) 또는 iOS(.ipa) 파일을 선택하세요. 또는 컴퓨터에서 파일을 드래그하여 콘솔에 드롭할 수도 있습니다.
3. 실행 이름을 입력하세요(예: **my first test**). 기본적으로 Device Farm 콘솔은 파일 이름을 사용합니다.
4. 다음을 선택하세요.
5. 구성 페이지의 테스트 프레임워크 설정 아래에서 테스트 프레임워크 또는 기본 제공 테스트 스위트 중 하나를 선택하세요. 각 옵션에 대한 자세한 내용은 테스트 유형 및 프레임워크 단원을 참조하세요.

- 아직 Device Farm용으로 테스트를 패키징하지 않은 경우 내장: Fuzz를 선택하여 기본 제공 테스트 스위트를 실행하십시오. 이벤트 수, 이벤트 스로틀 및 Randomizer 시드의 기본값을 유지할 수 있습니다. 자세한 내용은 the section called “내장: fuzz (Android 및 iOS)” 단원을 참조하세요.
- 지원되는 테스트 프레임워크 중 하나의 테스트 패키지가 있는 경우 해당 테스트 프레임워크를 선택한 다음 테스트가 포함된 파일을 업로드하십시오.

6. 다음을 선택하세요.
7. 디바이스 선택 페이지에서 디바이스 풀에 대해 상위 디바이스를 선택하세요.
8. 다음을 선택하세요.
9. 디바이스 상태 지정 페이지에서 다음 중 원하는 작업을 수행하세요.
   - 실행 중에 Device Farm에서 사용할 추가 데이터를 제공하려면 추가 데이터 추가에서.zip 파일을 업로드하세요.
   - 실행을 위해 다른 앱을 설치하려면 다른 앱 설치에서 해당 앱의 .apk 또는.ipa 파일을 업로드하세요. 설치 순서를 변경하려면 파일을 드래그 앤 드롭하세요.
   - 실행을 위해 Wi-Fi, Bluetooth, GPS 또는 NFC 라디오를 켜려면 라디오 상태 설정에서 해당 확인란을 선택하세요.

     > **Note**
     >
     > 현재 디바이스 무선 상태 설정은 Android 기본 테스트에만 사용 가능합니다.

   - 실행 중에 위치별 동작을 테스트하려면 디바이스 위치에서 사전 설정된 위도 및 경도 좌표를 지정하세요.
   - 실행에 사용할 디바이스 언어 및 지역을 사전 설정하려면 디바이스 로캘에서 로캘을 선택하세요.
   - 실행에 사용할 네트워크 프로필을 미리 설정하려면 네트워크 프로필에서 선별된 프로필을 선택하세요. 또는 네트워크 프로필 만들기를 선택하여 직접 만들 수도 있습니다.
10. 다음을 선택하세요.
11. 검토 및 실행 시작 페이지에서 확인 및 실행 시작을 선택하세요.

디바이스를 사용할 수 있게 되면 일반적으로 몇 분 이내에 Device Farm이 실행을 시작합니다. 실행 상태를 보려면 프로젝트의 자동 테스트 페이지에서 실행 이름을 선택하세요. 실행 페이지의 디바이스 아래에 있는 각 디바이스는 디바이스 테이블의 보류 중 아이콘

로

시작하다가 테스트가 시작되면 실행 중 아이콘

으로 전환됩니다. 각 테스트가 완료되면 콘솔에 디바이스 이름 옆에 테스트 결과 아이콘이 표시됩니다. 모든 테스트가 완료되면 실행 옆에 있는 보류 중 아이콘이 테스트 결과 아이콘으로 바뀝니다.

## 4단계: 실행 결과 보기

실행의 테스트 결과를 보려면 프로젝트의 자동 테스트 페이지에서 실행 이름을 선택하세요. 요약 페이지에 다음이 표시됩니다.

- 결과별 총 테스트 수
- 고유한 경고가 있거나 실패한 테스트 목록
- 각각에 대한 디바이스 및 테스트 결과 목록
- 디바이스별로 그룹화한 실행 중 캡처한 스크린샷
- 파싱 결과를 다운로드할 수 있는 단원

자세한 내용은 Device Farm에서 테스트 보고서 작업 단원을 참조하세요.

## 다음 단계

Device Farm에 관한 자세한 내용은 개념 단원을 참조하세요.

# Device Farm에서 디바이스 슬롯 구매

Device Farm 콘솔, AWS Command Line Interface (AWS CLI) 또는 Device Farm API를 사용하여 장치 슬롯을 구입할 수 있습니다.

주제



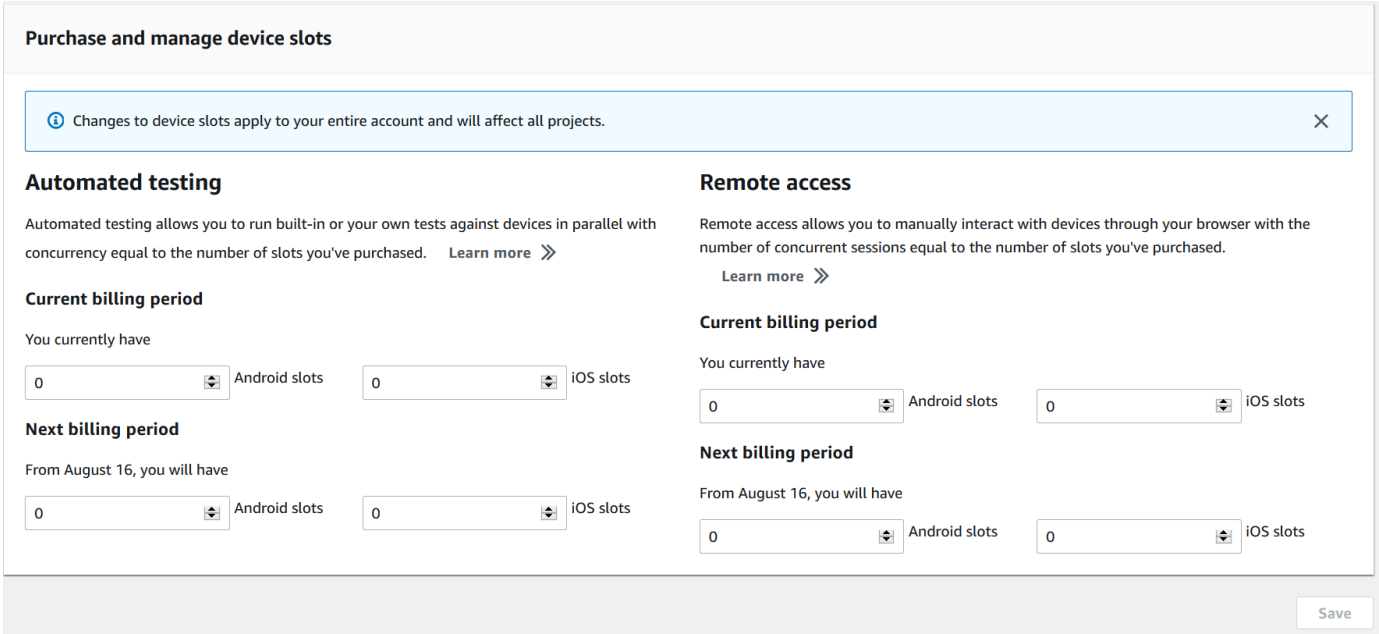
## 디바이스 슬롯 구입(콘솔)

1. https://console.aws.amazon.com/devicefarm에서 Device Farm 콘솔에 로그인하세요.
2. 탐색 창에서 모바일 디바이스 테스트를 선택하고 디바이스를 슬롯을 선택하세요.
3. 장치 슬롯 구매 및 관리 페이지에서 구매하려는 자동 테스트 및 원격 액세스 디바이스의 슬롯 수를 선택하여 사용자 지정 패키지를 만들 수 있습니다. 현재 청구 기간과 다음 청구 기간의 슬롯 금액을 모두 지정하세요.

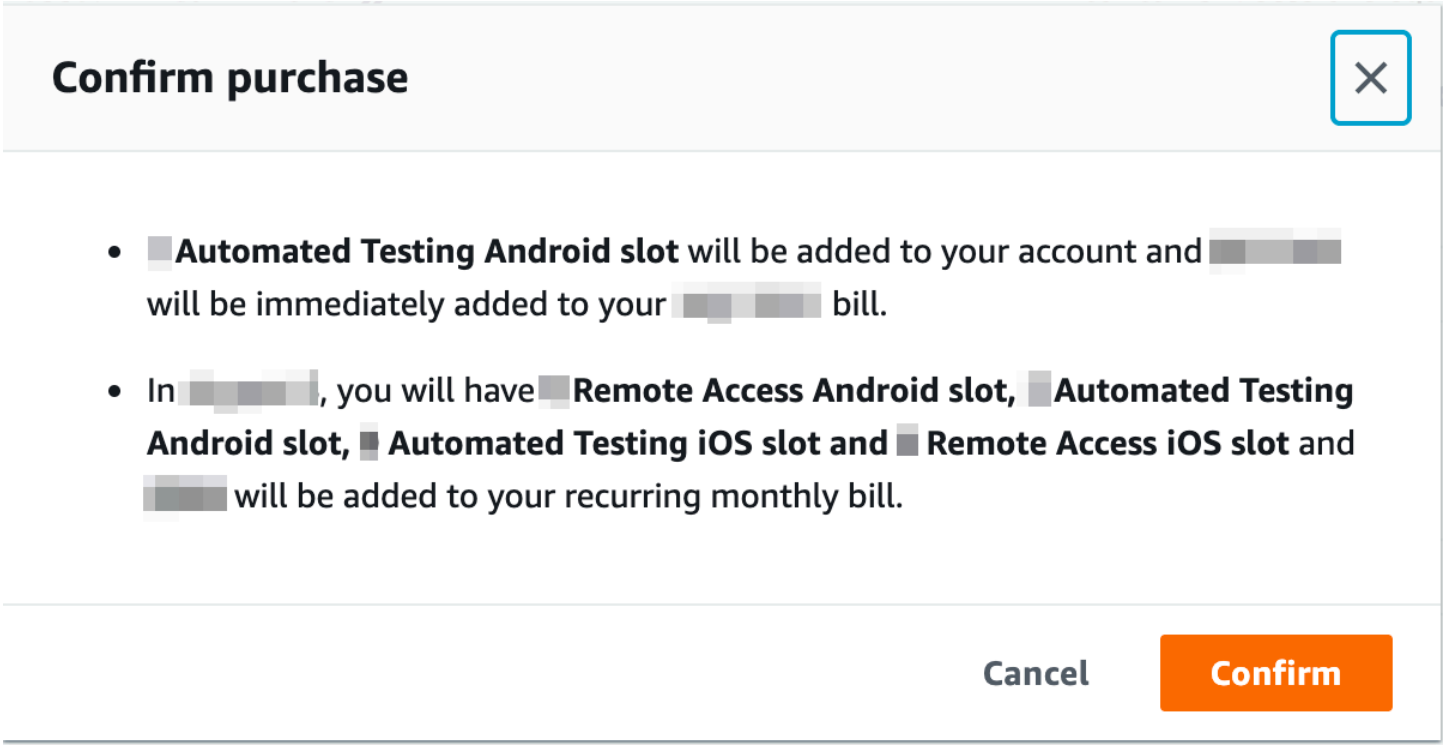
   슬롯 금액을 변경하면 텍스트가 청구 금액과 함께 동적으로 업데이트됩니다. 자세한 내용은 AWS Device Farm 요금을 참조하십시오.

   > **Important**
   >
   > 디바이스 슬롯 수를 변경했지만 AWS 연락처 또는 구매 문의 메시지를 참조한 경우, 요청하신 디바이스 슬롯 수만큼 구매가 아직 승인되지 않은 것입니다. AWS
   > 이 옵션을 선택하면 Device Farm 지원팀에 이메일을 보내라는 메시지가 표시됩니다. 이메일에서 구매하려는 각 장치 유형의 번호와 청구 주기를 지정하세요.

> **Note**
>
> 장치 슬롯에 대한 변경 사항은 전체 계정에 적용되며 모든 프로젝트에 영향을 미칩니다.




4. 구입을 선택하세요. 구매 확인 창이 나타납니다. 정보를 검토한 다음 확인을 선택하여 거래를 완료합니다.




디바이스 슬롯 구매 및 관리 페이지에서 현재 보유한 디바이스 슬롯 수를 확인할 수 있습니다. 슬롯 수를 늘리거나 줄인 경우 변경한 날로부터 한 달 후에 보유하게 될 슬롯 수가 표시됩니다.

## 디바이스 슬롯 구입(AWS CLI)

purchase-offering 명령을 실행하여 솔루션을 구입할 수 있습니다.

구입할 수 있는 최대 디바이스 슬롯 수와 남아 있는 무료 체험 시간(분) 등 Device Farm 계정 설정을 나열하려면 get-account-settings 명령을 실행하세요. 다음과 같은 결과가 출력됩니다.

```
{
    "accountSettings": {
        "maxSlots": {
            "GUID": 1,
            "GUID": 1,
            "GUID": 1,
            "GUID": 1
        },
        "unmeteredRemoteAccessDevices": {
            "ANDROID": 0,
            "IOS": 0
        },
        "maxJobTimeoutMinutes": 150,
```

```
        "trialMinutes": {
            "total": 1000.0,
            "remaining": 954.1
        },
        "defaultJobTimeoutMinutes": 150,
        "awsAccountNumber": "AWS-ACCOUNT-NUMBER",
        "unmeteredDevices": {
            "ANDROID": 0,
            "IOS": 0
        }
    }
}
```

사용 가능한 디바이스 슬롯 제품을 나열하려면 list-offerings 명령을 실행하세요. 다음과 유사한 출력 화면이 표시되어야 합니다.

```
{
    "offerings": [
        {
            "recurringCharges": [
                {
                    "cost": {
                        "amount": 250.0,
                        "currencyCode": "USD"
                    },
                    "frequency": "MONTHLY"
                }
            ],
            "platform": "IOS",
            "type": "RECURRING",
            "id": "GUID",
            "description": "iOS Unmetered Device Slot"
        },
        {
            "recurringCharges": [
                {
                    "cost": {
                        "amount": 250.0,
                        "currencyCode": "USD"
                    },
                    "frequency": "MONTHLY"
                }
            ],
```

```
            "platform": "ANDROID",
            "type": "RECURRING",
            "id": "GUID",
            "description": "Android Unmetered Device Slot"
        },
        {
            "recurringCharges": [
                {
                    "cost": {
                        "amount": 250.0,
                        "currencyCode": "USD"
                    },
                    "frequency": "MONTHLY"
                }
            ],
            "platform": "ANDROID",
            "type": "RECURRING",
            "id": "GUID",
            "description": "Android Remote Access Unmetered Device Slot"
        },
        {
            "recurringCharges": [
                {
                    "cost": {
                        "amount": 250.0,
                        "currencyCode": "USD"
                    },
                    "frequency": "MONTHLY"
                }
            ],
            "platform": "IOS",
            "type": "RECURRING",
            "id": "GUID",
            "description": "iOS Remote Access Unmetered Device Slot"
        }
    ]
}
```

사용 가능한 제품 프로모션을 나열하려면 list-offering-promotions 명령을 실행하세요.

> **Note**
>
> 이 명령은 아직 구입하지 않은 프로모션만 반환합니다. 프로모션을 통해 판매되는 제품 중에서 하나 이상의 슬롯을 구입하는 즉시 해당 프로모션은 더 이상 결과에 나타나지 않습니다.

다음과 유사한 출력 화면이 표시되어야 합니다.

```
{
    "offeringPromotions": [
        {
            "id": "2FREEMONTHS",
            "description": "New device slot customers get 3 months for the price of 1."
        }
    ]
}
```

제공 솔루션의 상태를 확인하려면 get-offering-status 명령을 실행하세요. 다음과 유사한 출력 화면이 표시되어야 합니다.

```
{
    "current": {
        "GUID": {
            "offering": {
                "platform": "IOS",
                "type": "RECURRING",
                "id": "GUID",
                "description": "iOS Unmetered Device Slot"
            },
            "quantity": 1
        },
        "GUID": {
            "offering": {
                "platform": "ANDROID",
                "type": "RECURRING",
                "id": "GUID",
                "description": "Android Unmetered Device Slot"
            },
            "quantity": 1
        }
    },
    "nextPeriod": {
```

```
        "GUID": {
            "effectiveOn": 1459468800.0,
            "offering": {
                "platform": "IOS",
                "type": "RECURRING",
                "id": "GUID",
                "description": "iOS Unmetered Device Slot"
            },
            "quantity": 1
        },
        "GUID": {
            "effectiveOn": 1459468800.0,
            "offering": {
                "platform": "ANDROID",
                "type": "RECURRING",
                "id": "GUID",
                "description": "Android Unmetered Device Slot"
            },
            "quantity": 1
        }
    }
}
```

이 기능에 renew-offering 및 list-offering-transactions 명령도 사용할 수 있습니다. 자세한 내용은 AWS CLI 참조을 참조하세요.

## 디바이스 슬롯 구입(API)

1. GetAccount설정 작업을 호출하여 계정 설정을 나열합니다.
2. ListOfferings오퍼레이션을 호출하여 사용 가능한 디바이스 슬롯 제품을 나열하십시오.
3. ListOffering프로모션 팀에 문의하여 이용 가능한 오퍼링 프로모션을 나열하십시오.

> **Note**
>
> 이 명령은 아직 구입하지 않은 프로모션만 반환합니다. 제품 프로모션을 통해 하나 이상의 슬롯을 구입하는 즉시 해당 프로모션은 더 이상 결과에 나타나지 않습니다.

4. 오퍼링을 PurchaseOffering구매하려면 오퍼레이션에 문의하세요.
5. GetOffering상태 오퍼레이션을 호출하여 오퍼링 상태를 확인하십시오.

이 기능에는 RenewOffering및 ListOffering트랜잭션 명령도 사용할 수 있습니다.

Device Farm API 사용에 대한 자세한 내용은 Device Farm 자동화 단원을 참조하세요.

## 디바이스 슬롯 취소 (콘솔)

1. https://console.aws.amazon.com/devicefarm에서 Device Farm 콘솔에 로그인하세요.
2. 탐색 창에서 모바일 디바이스 테스트를 선택하고 디바이스를 슬롯을 선택하세요.
3. 디바이스 슬롯 구매 및 관리 페이지에서 다음 결제 기간의 값을 줄여 자동 테스트와 원격 액세스를 위한 디바이스 슬롯 수를 줄일 수 있습니다. 다음 청구 주기에 계정에 청구되는 금액은 청구 기간 필드 아래에 나열됩니다.
4. 저장을 선택합니다. 변경 확인 창이 나타납니다. 정보를 검토한 다음 확인을 선택하여 거래를 완료합니다.

## 디바이스 슬롯 취소 (AWS CLI)

renew-offering명령을 실행하여 다음 청구 주기의 디바이스 수를 변경할 수 있습니다.

## 디바이스 슬롯 (API) 취소

RenewOffering오퍼레이션을 호출하여 계정 내 디바이스 수량을 변경하세요.

# AWS Device Farm 개념

이 단원에서는 중요한 Device Farm 개념에 대해 설명합니다.

- AWS Device Farm에서 디바이스 지원
- 테스트 환경
- 실행
- 앱
- AWS Device Farm에 있는 보고서
- 세션

Device Farm에서 지원되는 테스트 유형에 대한 자세한 내용은 AWS Device Farm에서 테스트 유형을 사용한 작업 단원을 참조하세요.

## AWS Device Farm에서 디바이스 지원

다음 단원에서는 Device Farm에 사용되는 디바이스 지원에 대한 정보를 제공합니다.

주제

- 지원되는 디바이스
- 디바이스 풀
- 프라이빗 디바이스
- 디바이스 브랜딩
- 디바이스 슬롯
- 사전 설치된 디바이스 앱
- 디바이스 기능

### 지원되는 디바이스

Device Farm은 수백 개의 고유하고 인기 있는 Android 및 iOS 디바이스와 운영 체제 조합을 지원합니다. 새 디바이스가 시장에 출시됨에 따라 사용 가능한 디바이스 목록이 늘어납니다. 전체 디바이스 목록은 디바이스 목록을 참조하세요.

## 디바이스 풀

Device Farm은 디바이스를 테스트에 사용할 수 있는 디바이스 풀로 구성합니다. 이러한 디바이스 풀에는 Android에서만 실행되거나 iOS에서만 실행되는 디바이스와 같은 관련 디바이스가 포함됩니다. Device Farm은 상위 디바이스용 풀과 같은 큐레이션된 디바이스 풀을 제공합니다. 공용 디바이스와 프라이빗 디바이스를 혼합하는 디바이스 풀을 만들 수도 있습니다.

## 프라이빗 디바이스

프라이빗 디바이스를 사용하면 테스트 요구 사항에 맞는 정확한 하드웨어 및 소프트웨어 구성을 지정할 수 있습니다. 루팅된 Android 디바이스와 같은 특정 구성은 프라이빗 디바이스로 지원될 수 있습니다. 프라이빗 디바이스란 Device Farm이 사용자를 대신하여 Amazon 데이터 센터에서 배포하는 물리적 모바일 디바이스입니다. 프라이빗 디바이스는 자동 테스트와 수동 테스트 모두에 한해 독점적으로 사용할 수 있습니다. 구독을 종료하기로 선택하면 하드웨어가 환경에서 제거됩니다. 자세한 내용은 프라이빗 디바이스와 AWS Device Farm에서 프라이빗 디바이스로 작업 단원을 참조하세요.

## 디바이스 브랜딩

Device Farm은 다양한 OEM의 실제 모바일 및 태블릿 디바이스에서 테스트를 실행합니다.

## 디바이스 슬롯

디바이스 슬롯은 동시성에 해당합니다. 구매한 디바이스 슬롯 수에 따라 테스트 또는 원격 액세스 세션에서 실행할 수 있는 디바이스 수가 결정됩니다.

디바이스 슬롯에는 두 가지 유형이 있습니다.

- 원격 액세스 디바이스 슬롯은 원격 액세스 세션에서 동시에 실행할 수 있는 슬롯입니다.

  원격 액세스 디바이스 슬롯이 하나인 경우 한 번에 하나의 원격 액세스 세션만 실행할 수 있습니다. 원격 테스트 디바이스 슬롯을 추가로 구입하면 여러 세션을 동시에 실행할 수 있습니다.
- 자동 테스트 디바이스 슬롯은 테스트를 동시에 실행할 수 있는 슬롯입니다.

  자동 테스트 디바이스 슬롯이 하나인 경우 한 번에 한 디바이스에서만 테스트를 실행할 수 있습니다. 자동 테스트 디바이스 슬롯을 추가로 구매하면 여러 디바이스에서 동시에 여러 테스트를 실행하여 테스트 결과를 더 빨리 얻을 수 있습니다.

디바이스 제품군에 따라(자동 테스트용 Android 또는 iOS 디바이스, 원격 액세스용 Android 또는 iOS 디바이스) 디바이스 슬롯을 구입할 수 있습니다. 자세한 내용은 Device Farm 요금을 참조하세요.

## 사전 설치된 디바이스 앱

Device Farm의 디바이스에는 제조업체와 이동통신사에서 이미 설치한 소수의 앱이 포함되어 있습니다.

## 디바이스 기능

모든 디바이스가 인터넷에 Wi-Fi로 연결되어 있습니다. 이동 통신사와 연결되어 있지 않으므로 전화를 걸거나 SMS 메시지를 보낼 수 없습니다.

전면 또는 후면 카메라를 지원하는 모든 디바이스로 사진을 찍을 수 있습니다. 디바이스를 장착하는 방식 때문에 사진이 어둡고 흐릿하게 보일 수 있습니다.

지원되는 디바이스에 Google Play 서비스가 설치되어 있지만 이러한 디바이스에는 활성 Google 계정이 없습니다.

# AWS Device Farm의 테스트 환경

AWS Device Farm은 자동 테스트를 실행하기 위한 사용자 지정 및 표준 테스트 환경을 모두 제공합니다. 자동 테스트를 완벽하게 제어할 수 있는 사용자 지정 테스트 환경을 선택할 수 있습니다. 또는 자동 테스트 스위트의 각 테스트에 대한 세분화된 보고를 제공하는 Device Farm 기본 표준 테스트 환경을 선택할 수 있습니다.

주제



## 표준 테스트 환경

표준 환경에서 테스트를 실행하면 Device Farm은 테스트 스위트의 모든 사례에 대한 자세한 로그 및 보고를 제공합니다. 각 테스트의 성능 데이터, 동영상, 스크린샷 및 로그를 확인하여 앱의 문제를 정확히 찾아내고 수정할 수 있습니다.

> **Note**
>
> Device Farm은 표준 환경에서 세분화된 보고 기능을 제공하므로 로컬에서 테스트를 실행할 때보다 테스트 실행 시간이 더 오래 걸릴 수 있습니다. 실행 시간을 단축하려면 사용자 지정 테스트 환경에서 테스트를 실행하세요.

## 사용자 지정 테스트 환경

테스트 환경을 사용자 지정할 때 Device Farm이 테스트를 수행하기 위해 실행해야 하는 명령을 지정할 수 있습니다. 이렇게 하면 Device Farm의 테스트가 로컬 시스템에서 실행되는 테스트와 유사한 방식으로 실행됩니다. 이 모드에서 테스트를 실행하면 테스트의 라이브 로그 및 비디오 스트리밍도 사용할 수 있습니다. 사용자 지정된 테스트 환경에서 테스트를 실행하면 각 테스트 사례에 대한 세분화된 보고서가 제공되지 않습니다. 자세한 내용은 사용자 지정 테스트 환경 작업 단원을 참조하세요.

Device Farm 콘솔, AWS CLI또는 Device Farm API를 사용하여 테스트 실행을 만들 때 사용자 지정 테스트 환경을 사용할 수 있습니다.

자세한 정보는 AWS CLI를 사용하여 사용자 지정 테스트 사양 업로드 및 Device Farm에서 테스트 실행 생성 단원을 참조하세요.

# AWS Device Farm에서 실행

다음 단원에는 Device Farm에서의 실행에 대한 정보가 있습니다.

Device Farm의 실행은 특정 디바이스 세트에서 실행될 특정 테스트 세트가 있는 앱의 특정 빌드입니다. 실행을 하면 결과에 대한 정보가 수록된 보고서가 생성됩니다. 실행에는 하나 이상의 작업이 있습니다.

주제

- 실행 구성
- 파일 보존 실행
- 디바이스 상태 실행
- 병렬 실행
- 실행 제한 시간 설정
- 실행 중 광고
- 실행 중 미디어
- 실행에 사용되는 일반적인 작업

## 실행 구성

실행의 일환으로 Device Farm이 현재 디바이스 설정을 재정의하는 데 사용할 수 있는 설정을 제공할 수 있습니다. 여기에는 위도 및 경도 좌표, 지역, 무선 상태(예: Bluetooth, GPS, NFC, Wi-Fi), 추가 데이터(.zip 파일에 포함됨) 및 보조 앱(앱을 테스트하기 전에 설치해야 하는 앱)이 포함됩니다.

## 파일 보존 실행

Device Farm은 앱과 파일을 30일 동안 저장한 다음 시스템에서 삭제합니다. 하지만 언제든 파일을 삭제할 수 있습니다.

Device Farm은 실행 결과, 로그 및 스크린샷을 400일 동안 저장한 다음 시스템에서 삭제합니다.

## 디바이스 상태 실행

Device Farm은 다음 작업에 사용할 수 있게 하기 전에 항상 디바이스를 재부팅합니다.

## 병렬 실행

Device Farm은 디바이스를 사용할 수 있게 되면 테스트를 병렬로 실행합니다.

## 실행 제한 시간 설정

각 디바이스의 테스트 실행을 중지하기 전에 테스트 실행이 지속되는 시간을 설정할 수 있습니다. 예를 들어 디바이스 하나당 테스트에 20분이 소요되는 경우 디바이스당 30분의 제한 시간을 선택해야 합니다.

자세한 정보는 AWS Device Farm의 테스트 실행 시간 제한 설정을 참조하세요.

## 실행 중 광고

Device Farm에 업로드하기 전에 앱에서 광고를 삭제하는 것이 좋습니다. 실행 중에는 광고가 표시된다고 보장할 수 없습니다.

## 실행 중 미디어

앱과 함께 사용할 미디어 또는 기타 데이터를 제공할 수 있습니다. 추가 데이터는 크기가 4GB를 넘지 않는 .zip 파일로 제공해야 합니다.

## 실행에 사용되는 일반적인 작업

자세한 정보는 Device Farm에서 테스트 실행 생성 및 AWS Device Farm에서 테스트 실행 작업 단원을 참조하세요.

# AWS Device Farm의 앱

다음 섹션에는 Device Farm의 앱 동작에 대한 정보가 포함되어 있습니다.

주제



## 계측 앱

앱을 계측하거나 Device Farm에 앱의 소스 코드를 제공할 필요가 없습니다. Android 앱은 수정하지 않고 제출할 수 있습니다. iOS 앱은 시뮬레이터 대신 iOS 디바이스 타겟으로 빌드해야 합니다.

## 실행 중인 앱 재서명

iOS 앱의 경우 프로비저닝 프로필에 Device Farm UUID를 추가할 필요가 없습니다. Device Farm은 내장된 프로비저닝 프로필을 와일드카드 프로필로 대체한 다음 앱을 재서명합니다. 보조 데이터를 제공하면 Device Farm이 보조 데이터를 설치하기 전에 Device Farm이 이를 앱 패키지에 추가하여 보조 데이터가 앱의 샌드박스에 존재하도록 합니다. 앱을 다시 서명하면 앱 그룹, 관련 도메인, 게임 센터,,, 무선 액세서리 구성, 인앱 구매 HealthKit HomeKit, 앱 간 오디오, Apple Pay, 푸시 알림, VPN 구성 및 제어와 같은 권한이 제거됩니다.

Android 앱의 경우 Device Farm은 앱을 재서명합니다. 이로 인해 Google Maps Android API와 같이 앱 서명에 의존하는 모든 기능이 중단되거나 다음과 같은 제품에서 불법 복제 방지 또는 변조 방지 기능이 실행될 수 있습니다. DexGuard

## 실행 시 난독화된 앱

Android 앱의 경우 앱이 난독화되더라도 Device Farm을 사용하여 테스트할 수 있습니다. ProGuard 하지만 불법 복제 방지 DexGuard 조치와 함께 사용하는 경우 Device Farm은 앱에 다시 서명하고 테스트를 실행할 수 없습니다.

# AWS Device Farm에 있는 보고서

다음 단원에서는 Device Farm 테스트 보고서에 대한 정보를 제공합니다.

주제

- 보고서의 로그
- 보고서의 일반적인 작업

## 보고서 보존

Device Farm은 보고서를 400일 동안 저장합니다. 이러한 보고서에는 메타데이터, 로그, 스크린샷 및 성능 데이터가 포함됩니다.

## 보고서 구성 요소

Device Farm의 보고서에는 통과 및 실패 정보, 충돌 보고서, 테스트 및 디바이스 로그, 스크린샷, 성능 데이터가 포함됩니다.

보고서에는 상세한 디바이스별 데이터와 높은 수준의 결과(예: 특정 문제의 발생 횟수)가 포함됩니다.

## 보고서의 로그

보고서에는 Android 테스트의 전체 logcat 캡처와 iOS 테스트의 전체 Device Console 로그가 포함됩니다.

## 보고서의 일반적인 작업

자세한 내용은 Device Farm에서 테스트 보고서 작업 단원을 참조하세요.

# AWS Device Farm에 사용되는 세션

Device Farm을 사용하면 웹 브라우저의 원격 액세스 세션을 통해 Android 및 iOS 앱의 대화형 테스트를 수행할 수 있습니다. 이러한 종류의 대화형 테스트를 통해 지원 엔지니어는 고객과 통화 시 고객의 문제를 단계별로 검토할 수 있습니다. 개발자는 특정 디바이스에서 문제를 재현하여 가능한 원인을 찾아낼 수 있습니다. 원격 세션을 사용하여 대상 고객을 대상으로 사용성 테스트를 수행할 수 있습니다.

주제

- 원격 액세스가 지원되는 디바이스
- 세션 파일 보존
- 계측 앱
- 세션 내 앱 재서명
- 세션 내 난독화된 앱

# 원격 액세스가 지원되는 디바이스

Device Farm은 고유하고 널리 사용되는 여러 Android 및 iOS 디바이스를 지원합니다. 새 디바이스가 시장에 출시됨에 따라 사용 가능한 디바이스 목록이 늘어납니다. Device Farm 콘솔에는 원격 액세스가 가능한 Android 및 iOS 디바이스의 현재 목록이 표시됩니다. 자세한 내용은 AWS Device Farm에서 디바이스 지원 단원을 참조하세요.

## 세션 파일 보존

Device Farm은 앱과 파일을 30일 동안 저장한 다음 시스템에서 삭제합니다. 하지만 언제든 파일을 삭제할 수 있습니다.

Device Farm은 세션 로그와 캡처한 비디오를 400일 동안 저장한 다음 시스템에서 삭제합니다.

## 계측 앱

앱을 계측하거나 Device Farm에 앱의 소스 코드를 제공할 필요가 없습니다. Android 및 iOS 앱은 수정하지 않고 제출할 수 있습니다.

## 세션 내 앱 재서명

Device Farm은 Android와 iOS 앱을 재서명합니다. 이로 인해 앱 서명에 의존하는 기능이 작동하지 않을 수 있습니다. 예를 들어 Android용 Google 지도 API는 앱의 서명에 따라 달라집니다. 앱 재서명은 Android 디바이스용 DexGuard와 같은 제품에서 불법 복제 방지 또는 변조 방지 탐지를 트리거할 수도 있습니다.

## 세션 내 난독화된 앱

Android 앱의 경우, 앱이 난독화되어 있더라도 ProGuard를 사용하면 Device Farm으로 테스트할 수 있습니다. 하지만 불법 복제 방지 조치와 함께 DexGuard를 사용하는 경우 Device Farm은 앱에 다시 서명할 수 없습니다.

# AWS Device Farm에서 프로젝트 작업

Device Farm의 프로젝트는 실행을 포함하는 Device Farm의 논리적 작업 영역을 나타내며, 하나 이상의 디바이스에서 단일 앱의 각 테스트당 한 번씩 실행됩니다. 프로젝트를 사용하면 원하는 방식으로 작업 영역을 조직할 수 있습니다. 예를 들어 앱 제목당 하나의 프로젝트 또는 플랫폼당 하나의 프로젝트가 있을 수 있습니다. 필요한 만큼 프로젝트를 생성할 수 있습니다.

AWS Device Farm 콘솔, AWS Command Line Interface(AWS CLI) 또는 AWS Device Farm API를 사용하여 프로젝트 작업을 수행할 수 있습니다.

주제



# AWS Device Farm에서 프로젝트 생성

AWS Device Farm 콘솔, AWS CLI 또는 AWS Device Farm API를 사용하여 프로젝트를 생성할 수 있습니다.

## 사전 조건

- 설정의 단계를 수행하세요.

## 프로젝트 생성(콘솔)

1. https://console.aws.amazon.com/devicefarm에서 Device Farm 콘솔에 로그인하세요.
2. Device Farm 탐색 패널에서 모바일 디바이스 테스트를 선택한 다음 프로젝트를 선택하세요.
3. 새 프로젝트를 선택하세요.
4. 프로젝트 이름을 입력한 다음 제출을 선택하세요.
5. 프로젝트에 대한 설정을 지정하려면 프로젝트 설정을 선택하세요. 이러한 설정에는 테스트 실행에 대한 기본 제한 시간이 포함됩니다. 설정이 적용되면 프로젝트의 모든 테스트 실행에 사용됩니다. 자세한 내용은 AWS Device Farm의 테스트 실행 시간 제한 설정 단원을 참조하세요.

## 프로젝트 생성(AWS CLI)

- 프로젝트 이름을 지정하여 create-project를 실행합니다.

  예제

  ```
  aws devicefarm create-project --name MyProjectName
  ```

  AWS CLI 응답에는 프로젝트의 Amazon 리소스 이름(ARN)이 포함되어 있습니다.

  ```
  {
      "project": {
          "name": "MyProjectName",
          "arn": "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
  c861-4c0a-b1d5-12345EXAMPLE",
          "created": 1535675814.414
      }
  }
  ```

  자세한 정보는 create-project 및 AWS CLI 참조 단원을 참조하세요.

## 프로젝트 생성(API)

- `CreateProject` API를 호출하세요.

Device Farm API 사용에 대한 자세한 내용은 Device Farm 자동화 단원을 참조하세요.

# AWS Device Farm에서 프로젝트 목록 보기

AWS Device Farm 콘솔, AWS CLI 또는 AWS Device Farm API를 사용하여 프로젝트 목록을 볼 수 있습니다.

주제

- 사전 조건
- 프로젝트 목록 보기(콘솔)
- 프로젝트 목록 보기(AWS CLI)
- 프로젝트 목록 보기(API)

## 사전 조건

- Device Farm에서 하나 이상의 프로젝트를 만드세요. AWS Device Farm에서 프로젝트 생성의 지침을 수행한 다음 이 페이지로 돌아오세요.

## 프로젝트 목록 보기(콘솔)

1. https://console.aws.amazon.com/devicefarm에서 Device Farm 콘솔에 로그인하세요.
2. 사용 가능한 프로젝트 목록을 찾으려면 다음을 수행하세요.
   - 모바일 디바이스 테스트 프로젝트의 경우 Device Farm 탐색 메뉴에서 모바일 디바이스 테스트를 선택한 다음 프로젝트를 선택하세요.
   - 데스크톱 브라우저 테스트 프로젝트의 경우 Device Farm 탐색 메뉴에서 데스크톱 브라우저 테스트를 선택한 다음 프로젝트를 선택하세요.

## 프로젝트 목록 보기(AWS CLI)

- 프로젝트 목록을 보려면 list-projects 명령을 실행합니다.

  단일 프로젝트에 대한 정보를 보려면 get-project 명령을 실행합니다.

AWS CLI의 Device Farm 사용 방법에 대한 자세한 내용은 AWS CLI 참조 단원을 참조하세요.

## 프로젝트 목록 보기(API)

- 프로젝트 목록을 보려면 `ListProjects` API를 호출하세요.

  단일 프로젝트에 대한 정보를 보려면 `GetProject` API를 호출하세요.

AWS Device Farm API에 대한 자세한 내용은 Device Farm 자동화 단원을 참조하세요.

# AWS Device Farm에서 테스트 실행 작업

Device Farm의 실행은 특정 디바이스 세트에서 실행될 특정 테스트 세트가 있는 앱의 특정 빌드입니다. 실행을 하면 결과에 대한 정보가 수록된 보고서가 생성됩니다. 실행에는 하나 이상의 작업이 있습니다. 자세한 정보는 실행을 참조하세요.

AWS Device Farm 콘솔, AWS Command Line Interface (AWS CLI) 또는 AWS Device Farm API를 사용하여 실행 작업을 수행할 수 있습니다.

주제



## Device Farm에서 테스트 실행 생성

Device Farm 콘솔 또는 Device Farm API를 사용하여 테스트 실행을 생성할 수 있습니다. AWS CLI또한 Device Farm용 Jenkins 또는 Gradle 플러그인 등 지원되는 플러그인을 사용할 수도 있습니다. 플러그인에 대한 자세한 내용은 도구 및 플러그인 단원을 참조하세요. 실행에 대한 자세한 내용은 실행 단원을 참조하세요.

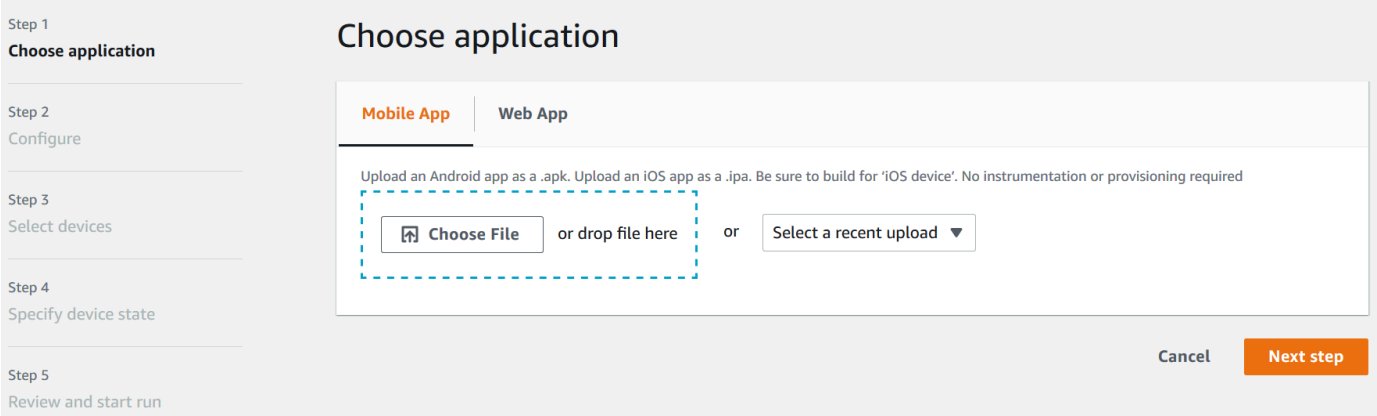
주제

# 사전 조건

Device Farm에 프로젝트가 있어야 합니다. AWS Device Farm에서 프로젝트 생성의 지침을 수행한 후 이 페이지로 돌아오세요.

## 테스트 실행 생성(콘솔)

1. https://console.aws.amazon.com/devicefarm에서 Device Farm 콘솔에 로그인하세요.
2. 탐색 창에서 모바일 디바이스 테스트를 선택하고 프로젝트를 선택하세요.
3. 프로젝트가 이미 있는 경우 해당 프로젝트에 테스트를 업로드할 수 있습니다. 그렇지 않다면 새 프로젝트를 선택하여 프로젝트의 이름을 입력한 뒤 생성을 선택하세요.
4. 프로젝트를 열고 새 실행 생성을 선택하세요.
5. 애플리케이션 선택 페이지에서 모바일 앱 또는 웹 앱을 선택하세요.




6. 애플리케이션 파일을 업로드하세요. 파일을 끌어서 놓거나 최근 업로드를 선택할 수도 있습니다. iOS 앱을 업로드하는 경우 시뮬레이터와 달리 iOS 디바이스를 선택해야 합니다.
7. (선택 사항) 실행 이름에 이름을 입력하세요. 기본적으로 Device Farm은 앱 파일 이름을 사용합니다.
8. 다음을 선택하세요.
9. 구성 페이지에서 사용 가능한 테스트 스위트 중 하나를 선택하세요.

> **Note**
>
> 사용 가능한 테스트가 없는 경우 내장: fuzz를 선택하여 기본 제공된 표준 테스트 스위트를 실행합니다. 내장: fuzz를 선택하고 이벤트 개수, 이벤트 스로틀, Radomizer 시드 상자가 표시되면 값을 변경하거나 유지할 수 있습니다.

사용할 수 있는 테스트 스위트에 대한 자세한 내용은 AWS Device Farm에서 테스트 유형을 사용한 작업 단원을 참조하세요.

10. 내장: fuzz를 선택하지 않은 경우 파일 선택을 선택한 후 자체 테스트가 포함된 파일을 찾아 선택하세요.
11. 실행 환경에서 표준 환경에서 테스트 실행 또는 테스트 환경 사용자 지정을 선택하세요. 자세한 내용은 테스트 환경 단원을 참조하세요.
12. 표준 테스트 환경을 사용하는 경우 13단계로 건너뛰세요. 기본 테스트 사양 YAML 파일이 있는 사용자 지정 테스트 환경을 사용하는 경우 13단계로 건너뛰세요.

    a. 사용자 지정 테스트 환경에서 기본 테스트 사양을 편집하려는 경우 편집을 선택하여 기본 YAML 사양을 업데이트하세요.

    b. 테스트 사양을 변경한 경우 새로 저장으로 테스트 사양을 업데이트하세요.
13. 동영상 녹화 또는 성능 데이터 캡처 옵션을 구성하려면 고급 구성을 선택하세요.

    a. 테스트 실행 중에 동영상 녹화하려면 동영상 녹화 활성화를 선택하세요.

    b. 디바이스에서 성능 데이터 캡처하려면 앱 성능 데이터 캡처 활성화를 선택하세요.

> **Note**
>
> 프라이빗 디바이스가 있는 경우 프라이빗 디바이스 구성도 표시됩니다.

14. 다음을 선택하세요.
15. 디바이스 선택 페이지에서 다음 중 하나를 수행하세요.

    - 테스트를 실행할 기본 제공 디바이스 풀을 선택하려면 디바이스 풀에서 최고 인기 디바이스를 선택하세요.
    - 테스트를 실행할 고유한 디바이스 풀을 만들려면 디바이스 풀 생성의 지침을 수행한 후 현재 페이지로 돌아오세요.
    - 이전에 고유한 디바이스 풀을 만들어 두었다면 디바이스 풀에서 해당 디바이스 풀을 선택하세요.

    자세한 내용은 AWS Device Farm에서 디바이스 지원 단원을 참조하세요.
16. 다음을 선택하세요.

17. 디바이스 상태 지정 페이지에서 다음을 수행하세요.

    - 실행 중에 Device Farm이 사용할 다른 데이터를 입력하려면 데이터 추가 옆의 파일 선택 클릭 후, 해당 데이터가 포함된 .zip 파일을 찾아 선택하세요.
    - 실행 중 Device Farm에 사용할 추가 앱을 설치하려면 다른 앱 설치 옆의 파일 선택 클릭 후 해당 앱이 들어 있는 .apk 또는 .ipa 파일을 찾아 선택하세요. 설치할 다른 앱에 대해서도 이 절차를 반복하세요. 앱을 업로드한 후 끌어서 놓는 방법으로 설치 순서를 변경할 수 있습니다.
    - 실행 중에 Wi-Fi, Bluetooth, GPS 또는 NFC 활성화 여부를 지정하려면 무선 상태 설정 옆의 알맞은 확인란을 선택하세요.
    - 실행을 위해 디바이스 위도 및 경도를 미리 설정하려면 디바이스 위치 옆에 좌표를 입력합니다.
    - 실행을 위해 디바이스 로캘을 미리 설정하려면 디바이스 로캘에서 로캘을 선택하세요.
18. 다음을 선택하세요.
19. 검토 및 실행 시작 페이지에서 테스트 실행 시간 제한을 지정할 수 있습니다. 무제한 테스트 슬롯을 사용하는 경우 과금되지 않는 슬롯에서 실행이 선택되어 있는지 확인하세요.
20. 값을 입력하거나 막대 슬라이더를 사용하여 실행 시간 제한을 변경하세요. 자세한 내용은 AWS Device Farm의 테스트 실행 시간 제한 설정 단원을 참조하세요.
21. 확인 및 실행 시작을 선택하세요.

디바이스를 사용할 수 있게 되면 일반적으로 몇 분 이내에 Device Farm이 실행을 시작합니다. 테스트 실행 중에 Device Farm 콘솔은 실행 테이블에 보류 중 아이콘  을 표시합니다. 실행 중인 각 디바이스 역시 보류 중 아이콘으로 시작하여 테스트가 시작되면 실행 중 아이콘  로 전환합니다. 각 테스트가 완료되면 디바이스 이름 옆에 테스트 결과 아이콘이 표시됩니다. 모든 테스트가 완료되면 실행 옆에 있는 보류 중 아이콘이 테스트 결과 아이콘으로 바뀝니다.

테스트 실행을 중지해야 하는 경우 AWS Device Farm에서의 실행 중지 단원을 참조하세요.

## 테스트 실행 생성(AWS CLI)

를 AWS CLI 사용하여 테스트 실행을 만들 수 있습니다.

주제

- 1단계: 프로젝트 선택

## 1단계: 프로젝트 선택

테스트 실행을 Device Farm 프로젝트와 연결해야 합니다.

1. Device Farm 프로젝트를 나열하려면 list-projects를 실행하세요. 프로젝트가 없는 경우 AWS Device Farm에서 프로젝트 생성를 참조하세요.

   예제

   ```
   aws devicefarm list-projects
   ```

   응답은 Device Farm 프로젝트 목록을 포함합니다.

   ```
   {
       "projects": [
           {
               "name": "MyProject",
               "arn": "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
   c861-4c0a-b1d5-12345EXAMPLE",
               "created": 1503612890.057
           }
       ]
   }
   ```

2. 테스트 실행과 연결할 프로젝트를 선택한 후 Amazon 리소스 이름(ARN)을 적어 둡니다.

## 2단계: 디바이스 풀 선택

테스트 실행과 연결할 디바이스 풀을 선택해야 합니다.

1. 디바이스 풀을 보려면 프로젝트 ARN을 지정하여 list-device-pools를 실행하세요.

   예제

```
aws devicefarm list-device-pools --arn arn:MyProjectARN
```

응답은 Top Devices 및 해당 프로젝트에 대해 기존 생성한 모든 디바이스 풀 등 내장 Device Farm 디바이스 풀을 포함합니다.

```
{
    "devicePools": [
        {
            "rules": [
                {
                    "attribute": "ARN",
                    "operator": "IN",
                    "value": "[\"arn:aws:devicefarm:us-west-2::device:example1\",
\"arn:aws:devicefarm:us-west-2::device:example2\",\"arn:aws:devicefarm:us-
west-2::device:example3\"]"
                }
            ],
            "type": "CURATED",
            "name": "Top Devices",
            "arn": "arn:aws:devicefarm:us-west-2::devicepool:example",
            "description": "Top devices"
        },
        {
            "rules": [
                {
                    "attribute": "PLATFORM",
                    "operator": "EQUALS",
                    "value": "\"ANDROID\""
                }
            ],
            "type": "PRIVATE",
            "name": "MyAndroidDevices",
            "arn": "arn:aws:devicefarm:us-west-2:605403973111:devicepool:example2"
        }
    ]
}
```

2. 디바이스 풀을 선택하고 ARN을 적어 둡니다

디바이스 풀을 생성한 후 이 단계로 돌아올 수도 있습니다. 자세한 내용은 디바이스 풀(AWS CLI)을 생성하세요. 단원을 참조하세요.

## 3단계: 애플리케이션 파일 업로드

업로드 요청을 생성하고 Amazon Simple Storage Service(Amazon S3)의 미리 서명된 업로드 URL을 가져오려면 다음이 필요합니다.

- 프로젝트 ARN
- 앱 파일의 이름
- 업로드 유형

자세한 정보는 create-upload을 참조하세요.

1. 파일을 업로드하려면 `--project-arn`, `--name` 및 `--type` 파라미터와 함께 create-upload을 실행하세요.

   다음 예시에서는 Android 앱의 업로드를 생성합니다.

   ```
   aws devicefarm create-upload --project-arn arn:MyProjectArn --name MyAndroid.apk --
   type ANDROID_APP
   ```

   응답에는 앱 업로드 ARN 및 미리 서명된 URL이 포함됩니다.

   ```
   {
       "upload": {
           "status": "INITIALIZED",
           "name": "MyAndroid.apk",
           "created": 1535732625.964,
           "url": "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/
   ExampleURL",
           "type": "ANDROID_APP",
           "arn": "arn:aws:devicefarm:us-west-2:123456789101:upload:5e01a8c7-
   c861-4c0a-b1d5-12345EXAMPLE"
       }
   }
   ```

2. 앱 업로드 ARN 및 미리 서명된 URL을 적어 둡니다

3. Amazon S3의 미리 서명된 URL을 사용하여 앱 파일을 업로드하세요. 다음 예에서는 curl을 사용하여 Android .apk 파일을 업로드합니다.

```
curl -T MyAndroid.apk "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/
ExampleURL"
```

자세한 내용을 알아보려면 Amazon Simple Storage Service 사용 설명서의 미리 서명된 URL을 사용하여 객체 업로드를 참조하세요.

4. 앱 업로드 상태를 확인하려면 get-upload를 실행하고 앱 업로드의 ARN을 지정하세요.

```
aws devicefarm get-upload --arn arn:MyAppUploadARN
```

응답의 상태가 SUCCEEDED 될 때까지 기다린 후 테스트 스크립트 패키지를 업로드하세요.

```
{
    "upload": {
        "status": "SUCCEEDED",
        "name": "MyAndroid.apk",
        "created": 1535732625.964,
        "url": "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/
ExampleURL",
        "type": "ANDROID_APP",
        "arn": "arn:aws:devicefarm:us-west-2:123456789101:upload:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE",
        "metadata": "{"valid": true}"
    }
}
```

## 4단계: 테스트 스크립트 패키지 업로드

다음으로 테스트 스크립트 패키지를 업로드하세요.

1. 업로드 요청을 생성하고 Amazon S3의 미리 서명된 업로드 URL을 가져오려면 `--project-arn`, `--name`, `--type` 파라미터와 함께 create-upload를 실행하세요.

   다음 예에서는 Appium Java TestNG 테스트 패키지 업로드를 생성합니다.

```
aws devicefarm create-upload --project-arn arn:MyProjectARN --name MyTests.zip --
type APPIUM_JAVA_TESTNG_TEST_PACKAGE
```

   응답에는 테스트 패키지 업로드 ARN 및 미리 서명된 URL이 포함됩니다.

```
{
    "upload": {
        "status": "INITIALIZED",
        "name": "MyTests.zip",
        "created": 1535738627.195,
        "url": "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/
ExampleURL",
        "type": "APPIUM_JAVA_TESTNG_TEST_PACKAGE",
        "arn": "arn:aws:devicefarm:us-west-2:123456789101:upload:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE"
    }
}
```

2. 테스트 패키지 업로드의 ARN과 미리 서명된 URL을 적어 둡니다

3. Amazon S3의 미리 서명된 URL을 사용하여 테스트 스크립트 패키지 파일을 업로드하세요. 다음 예에서는 curl을 사용하여 압축된 Appium TestNG 스크립트 파일을 업로드합니다.

```
curl -T MyTests.zip "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/
ExampleURL"
```

4. 테스트 스크립트 패키지 업로드의 상태를 확인하려면 get-upload를 실행하고 1단계의 테스트 패키지 업로드의 ARN을 지정하세요.

```
aws devicefarm get-upload --arn arn:MyTestsUploadARN
```

응답의 상태가 SUCCEEDED 될 때까지 기다린 후 선택 사항인 다음 단계를 진행하세요.

```
{
    "upload": {
        "status": "SUCCEEDED",
        "name": "MyTests.zip",
        "created": 1535738627.195,
        "url": "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/
ExampleURL",
        "type": "APPIUM_JAVA_TESTNG_TEST_PACKAGE",
        "arn": "arn:aws:devicefarm:us-west-2:123456789101:upload:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE",
        "metadata": "{"valid": true}"
    }
}
```

## 5단계: 사용자 지정 테스트 사양 업로드(선택 사항)

표준 테스트 환경에서 테스트를 실행하는 경우 이 단계를 건너뛰세요.

Device Farm은 지원되는 각 테스트 유형의 기본 테스트 사양 파일을 유지합니다. 다음으로 기본 테스트 사양을 다운로드하여 사용자 지정 테스트 환경에서 테스트를 실행하기 위해 사용자 지정 테스트 사양 업로드를 생성하세요. 자세한 내용은 테스트 환경 단원을 참조하세요.

1. 기본 테스트 사양의 업로드 ARN을 찾으려면 list-uploads를 실행하고 프로젝트 ARN을 지정하세요.

   ```
   aws devicefarm list-uploads --arn arn:MyProjectARN
   ```

   응답에 기본 테스트 사양의 각 항목이 포함됩니다.

   ```
   {
       "uploads": [
           {

               {
                   "status": "SUCCEEDED",
                   "name": "Default TestSpec for Android Appium Java TestNG",
                   "created": 1529498177.474,
                   "url": "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/
   ExampleURL",
                   "type": "APPIUM_JAVA_TESTNG_TEST_SPEC",
                   "arn": "arn:aws:devicefarm:us-west-2:123456789101:upload:5e01a8c7-
   c861-4c0a-b1d5-12345EXAMPLE"
               }
           }
       ]
   }
   ```

2. 목록에서 기본 테스트 사양을 선택하세요. 업로드 ARN을 적어 둡니다
3. 기본 테스트 사양을 다운로드하려면 get-upload를 실행하고 업로드 ARN을 지정하세요.

   예제

   ```
   aws devicefarm get-upload --arn arn:MyDefaultTestSpecARN
   ```

   응답은 기본 테스트 사양을 다운로드할 수 있는 미리 서명된 URL을 포함합니다.

4. 다음 예에서는 curl을 사용하여 기본 테스트 사양을 다운로드하고 MyTestSpec.yml로 저장합니다.

```
curl "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/ExampleURL" >
 MyTestSpec.yml
```

5. 테스트 요구 사항에 맞게 기본 테스트 사양을 편집한 후, 향후 테스트 실행에서 수정된 테스트 사양을 사용할 수 있습니다. 사용자 지정 테스트 환경에서 기본 테스트 사양을 그대로 사용하려면 이 단계를 건너뛰세요.

6. 사용자 지정 테스트 사양의 업로드를 생성하려면 테스트 사양 이름, 테스트 사양 유형 및 프로젝트 ARN을 지정하여 create-upload를 실행하세요.

   다음 예에서는 Appium Java TestNG 사용자 지정 테스트 사양 업로드를 생성합니다.

```
aws devicefarm create-upload --name MyTestSpec.yml --type
 APPIUM_JAVA_TESTNG_TEST_SPEC --project-arn arn:MyProjectARN
```

   응답에는 테스트 사양 업로드 ARN 및 미리 서명된 URL이 포함됩니다.

```
{
    "upload": {
        "status": "INITIALIZED",
        "category": "PRIVATE",
        "name": "MyTestSpec.yml",
        "created": 1535751101.221,
        "url": "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/
ExampleURL",
        "type": "APPIUM_JAVA_TESTNG_TEST_SPEC",
        "arn": "arn:aws:devicefarm:us-west-2:123456789101:upload:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE"
    }
}
```

7. 테스트 사양 업로드의 ARN과 미리 서명된 URL을 적어 둡니다

8. Amazon S3의 미리 서명된 URL을 사용하여 테스트 사양 파일을 업로드하세요. 이 예제는 Appium JavaTest NG 테스트 사양을 curl 업로드하는 데 사용됩니다.

```
curl -T MyTestSpec.yml "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/
ExampleURL"
```

9. 테스트 사양 업로드의 상태를 확인하려면 get-upload를 실행하고 업로드 ARN을 지정하세요.

```
aws devicefarm get-upload --arn arn:MyTestSpecUploadARN
```

응답의 상태가 SUCCEEDED 될 때까지 기다린 후 테스트 실행을 예약하세요.

```
{
    "upload": {
        "status": "SUCCEEDED",
        "name": "MyTestSpec.yml",
        "created": 1535732625.964,
        "url": "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/
ExampleURL",
        "type": "APPIUM_JAVA_TESTNG_TEST_SPEC",
        "arn": "arn:aws:devicefarm:us-west-2:123456789101:upload:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE",
        "metadata": "{"valid": true}"
    }
}
```

사용자 지정 테스트 사양을 업데이트하려면 테스트 사양의 업로드 ARN을 지정하여 update-upload를 실행하세요. 자세한 정보는 update-upload을 참조하세요.

## 6단계: 테스트 실행 예약

를 사용하여 테스트 실행을 예약하려면 다음을 지정하여 AWS CLI실행합니다schedule-run.

- 1단계의 프로젝트 ARN
- 2단계의 디바이스 풀 ARN
- 3단계의 앱 업로드 ARN
- 4단계의 테스트 패키지 업로드 ARN

사용자 지정 테스트 환경에서 테스트를 실행하는 경우 5단계의 테스트 사양 ARN도 필요합니다.

표준 테스트 환경에서 실행을 예약

- 프로젝트 ARN, 디바이스 풀 ARN, 애플리케이션 업로드 ARN 및 테스트 패키지 정보를 지정하여 schedule-run을 실행하세요.

예제

```
aws devicefarm schedule-run --project-arn arn:MyProjectARN --app-
arn arn:MyAppUploadARN --device-pool-arn arn:MyDevicePoolARN --name MyTestRun --
test type=APPIUM_JAVA_TESTNG,testPackageArn=arn:MyTestPackageARN
```

응답에는 테스트 실행 상태를 확인할 때 사용할 수 있는 실행 ARN이 포함됩니다.

```
{
    "run": {
        "status": "SCHEDULING",
        "appUpload": "arn:aws:devicefarm:us-west-2:123456789101:upload:5e01a8c7-
c861-4c0a-b1d5-12345appEXAMPLE",
        "name": "MyTestRun",
        "radios": {
            "gps": true,
            "wifi": true,
            "nfc": true,
            "bluetooth": true
        },
        "created": 1535756712.946,
        "totalJobs": 179,
        "completedJobs": 0,
        "platform": "ANDROID_APP",
        "result": "PENDING",
        "devicePoolArn": "arn:aws:devicefarm:us-
west-2:123456789101:devicepool:5e01a8c7-c861-4c0a-b1d5-12345devicepoolEXAMPLE",
        "jobTimeoutMinutes": 150,
        "billingMethod": "METERED",
        "type": "APPIUM_JAVA_TESTNG",
        "testSpecArn": "arn:aws:devicefarm:us-west-2:123456789101:upload:5e01a8c7-
c861-4c0a-b1d5-12345specEXAMPLE",
        "arn": "arn:aws:devicefarm:us-west-2:123456789101:run:5e01a8c7-c861-4c0a-
b1d5-12345runEXAMPLE",
        "counters": {
            "skipped": 0,
            "warned": 0,
            "failed": 0,
            "stopped": 0,
            "passed": 0,
            "errored": 0,
            "total": 0
```

```
        }
    }
}
```

자세한 정보는 schedule-run을 참조하세요.

사용자 지정 테스트 환경에서 실행을 예약

- 단계는 `--test` 파라미터의 추가 `testSpecArn` 속성이 포함된 표준 테스트 환경의 단계와 거의 동일합니다.

  예제

```
aws devicefarm schedule-run --project-arn arn:MyProjectARN --app-
arn arn:MyAppUploadARN --device-pool-arn arn:MyDevicePoolARN --name MyTestRun --
test
 testSpecArn=arn:MyTestSpecUploadARN,type=APPIUM_JAVA_TESTNG,testPackageArn=arn:MyTestPacka
```

테스트 실행 상태를 확인

- get-run 명령을 사용하고 실행 ARN을 지정하세요.

```
aws devicefarm get-run --arn arn:aws:devicefarm:us-
west-2:111122223333:run:5e01a8c7-c861-4c0a-b1d5-12345runEXAMPLE
```

자세한 정보는 get-run을 참조하세요. Device Farm을 와 함께 사용하는 방법에 대한 자세한 내용은 을 AWS CLI참조하십시오AWS CLI 참조.

## 테스트 실행 생성(API)

단계는 AWS CLI 섹션에 설명된 단계와 동일합니다. 테스트 실행 생성(AWS CLI) 단원을 참조하세요.

`ScheduleRun` API를 호출하려면 다음 정보가 필요합니다.

- 프로젝트 ARN 프로젝트 생성(API) 및 `CreateProject` 단원을 참조하세요.
- 애플리케이션 업로드 ARN `CreateUpload` 단원을 참조하세요.
- 테스트 패키지 업로드 ARN `CreateUpload` 단원을 참조하세요.
- 디바이스 풀 ARN 디바이스 풀 생성 및 `CreateDevicePool`을 참조하세요.

> **Note**
>
> 사용자 지정 테스트 환경에서 테스트를 실행하는 경우 테스트 사양 업로드 ARN도 필요합니다. 자세한 내용은 5단계: 사용자 지정 테스트 사양 업로드(선택 사항) 및 `CreateUpload` 섹션을 참조하세요.

자세한 Device Farm API 사용은 Device Farm 자동화 단원을 참조하세요.

## 다음 단계

Device Farm 콘솔에서 실행이 완료되면 시계 아이콘

이

성공

등의 결과 아이콘으로 바뀝니다. 테스트가 완료되자마자 실행에 대한 보고서가 표시됩니다. 자세한 내용은 AWS Device Farm에 있는 보고서 단원을 참조하세요.

보고서를 사용하려면 Device Farm에서 테스트 보고서 작업의 지침을 따르세요.

# AWS Device Farm의 테스트 실행 시간 제한 설정

각 디바이스의 테스트 실행을 중지하기 전에 테스트 실행이 지속되는 시간을 설정할 수 있습니다. 기본 실행 시간 제한은 디바이스당 150분이지만 최소 5분까지 적은 값을 설정할 수 있습니다. AWS Device Farm 콘솔 또는 AWS Device Farm API를 사용하여 실행 제한 시간을 설정할 수 있습니다. AWS CLI

> **Important**
>
> 실행 시간 제한 옵션은 일부 버퍼와 함께 테스트 실행에 대해 최대 시간으로 설정되어야 합니다. 예를 들어 디바이스 하나당 테스트에 20분이 소요되는 경우 디바이스당 30분의 시간 제한을 선택해야 합니다.

실행이 시간 제한을 초과하면 해당 디바이스에서 실행이 강제로 중지됩니다. 가능한 경우 부분적인 결과가 제공됩니다. 과금 청구 옵션을 사용하는 경우 그 시점까지의 실행에 대한 비용이 청구됩니다. 요금에 대한 자세한 내용은 Device Farm 요금을 참조하세요.

각 디바이스에서 테스트 실행이 얼마나 걸리는지 알고 있다면 이 기능을 사용할 수 있습니다. 테스트 실행에 실행 시간 제한을 지정하면 테스트 실행이 중단되는 상황에서 진행이 멈춘 상태의 시간에 대한 요금이 디바이스에 청구되는 것을 피할 수 있습니다. 즉, 실행 시간 제한 기능을 사용하면 예상보다 실행이 오래 걸릴 경우 실행을 중지할 수 있습니다.

프로젝트 레벨 그리고 테스트 실행 레벨 두 군데에서 실행 시간 제한을 설정할 수 있습니다.

## 사전 조건

1. 설정의 단계를 수행하세요.
2. Device Farm에서 프로젝트를 생성하세요. AWS Device Farm에서 프로젝트 생성의 지침을 수행한 후 해당 페이지로 돌아오세요.

## 프로젝트의 실행 시간 제한 설정

1. https://console.aws.amazon.com/devicefarm 에서 Device Farm 콘솔에 로그인하세요.
2. Device Farm 탐색 패널에서 모바일 디바이스 테스트를 선택한 다음 프로젝트를 선택하세요.
3. 프로젝트가 이미 있는 리스트에서 해당 프로젝트를 선택하세요. 그렇지 않으면 새 프로젝트를 선택하여 프로젝트 이름을 입력한 후 제출을 선택하세요.
4. 프로젝트 설정을 선택하세요.
5. 일반 탭의 실행 시간 제한에 값을 입력하거나 막대 슬라이더를 사용합니다.
6. 저장을 선택하세요.

   실행을 예약할 때 시간 제한 값을 재정의하지 않는 한 프로젝트의 모든 테스트 실행에 방금 지정한 실행 시간 제한 값이 사용됩니다.

## 테스트 실행을 위한 실행 시간 제한 설정

1. https://console.aws.amazon.com/devicefarm에서 Device Farm 콘솔에 로그인하세요.
2. Device Farm 탐색 패널에서 모바일 디바이스 테스트를 선택한 다음 프로젝트를 선택하세요.
3. 프로젝트가 이미 있는 리스트에서 해당 프로젝트를 선택하세요. 그렇지 않으면 새 프로젝트를 선택하여 프로젝트 이름을 입력한 후 제출을 선택하세요.
4. 새 실행 생성을 선택하세요.
5. 단계에 따라 애플리케이션을 선택하고 테스트를 구성한 후, 디바이스를 선택하여 상태를 지정하세요.

6. 검토 및 실행 시작의 실행 시간 제한에 값을 입력하거나 막대 슬라이더를 사용하세요.
7. 확인 및 실행 시작을 선택하세요.

# AWS Device Farm 실행을 위한 네트워크 연결 및 조건 시뮬레이션

Device Farm에서 Android, iOS, FireOS 및 웹 앱을 테스트하는 동안 네트워크 쉐이핑을 사용하여 네트워크 연결 및 조건을 시뮬레이션할 수 있습니다. 예를 들어, 완벽하지 않은 네트워크 조건에서 앱을 테스트할 수 있습니다.

기본 네트워크 설정을 사용하여 실행을 생성하면 각 디바이스는 인터넷 연결을 통해 방해 없이 완전한 Wi-Fi 연결을 사용할 수 있습니다. 네트워크 셰이핑을 사용하는 경우 인바운드 및 아웃바운드 트래픽의 처리량, 지연, 지터 및 손실을 WiFi 제어하는 3G 또는 Lossy와 같은 네트워크 프로필을 지정하도록 Wi-Fi 연결을 변경할 수 있습니다.

주제

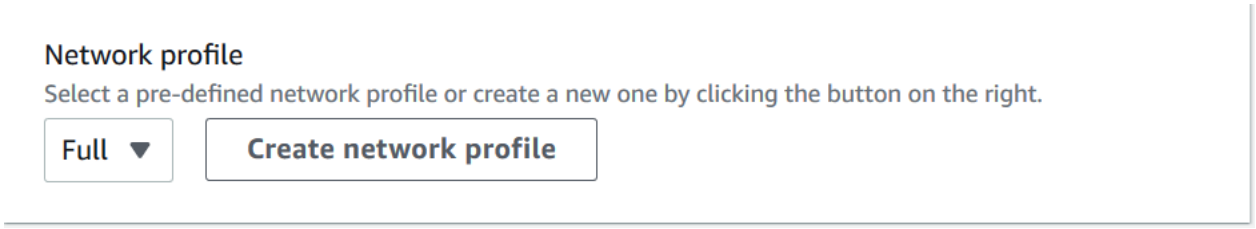
- 테스트 실행을 예약 시 네트워크 쉐이핑 설정
- 네트워크 프로파일 생성
- 테스트 중 네트워크 상태 변경

## 테스트 실행을 예약 시 네트워크 쉐이핑 설정

달리기를 예약할 때 Device Farm에서 큐레이팅한 프로필 중 하나를 선택하거나 직접 프로필을 만들고 관리할 수 있습니다.

1. 모든 Device Farm 프로젝트에서 새 실행 생성을 선택하세요.

   프로젝트가 없는 경우 AWS Device Farm에서 프로젝트 생성 단원을 참조하세요.
2. 애플리케이션을 선택한 후 다음을 선택하세요.
3. 테스트를 구성한 후 다음을 선택하세요.
4. 디바이스를 선택하고 다음을 선택하세요.
5. 위치 및 네트워크 설정 섹션에서 네트워크 프로필을 선택하거나 네트워크 프로필 생성을 선택하여 고유한 프로필을 생성하세요.

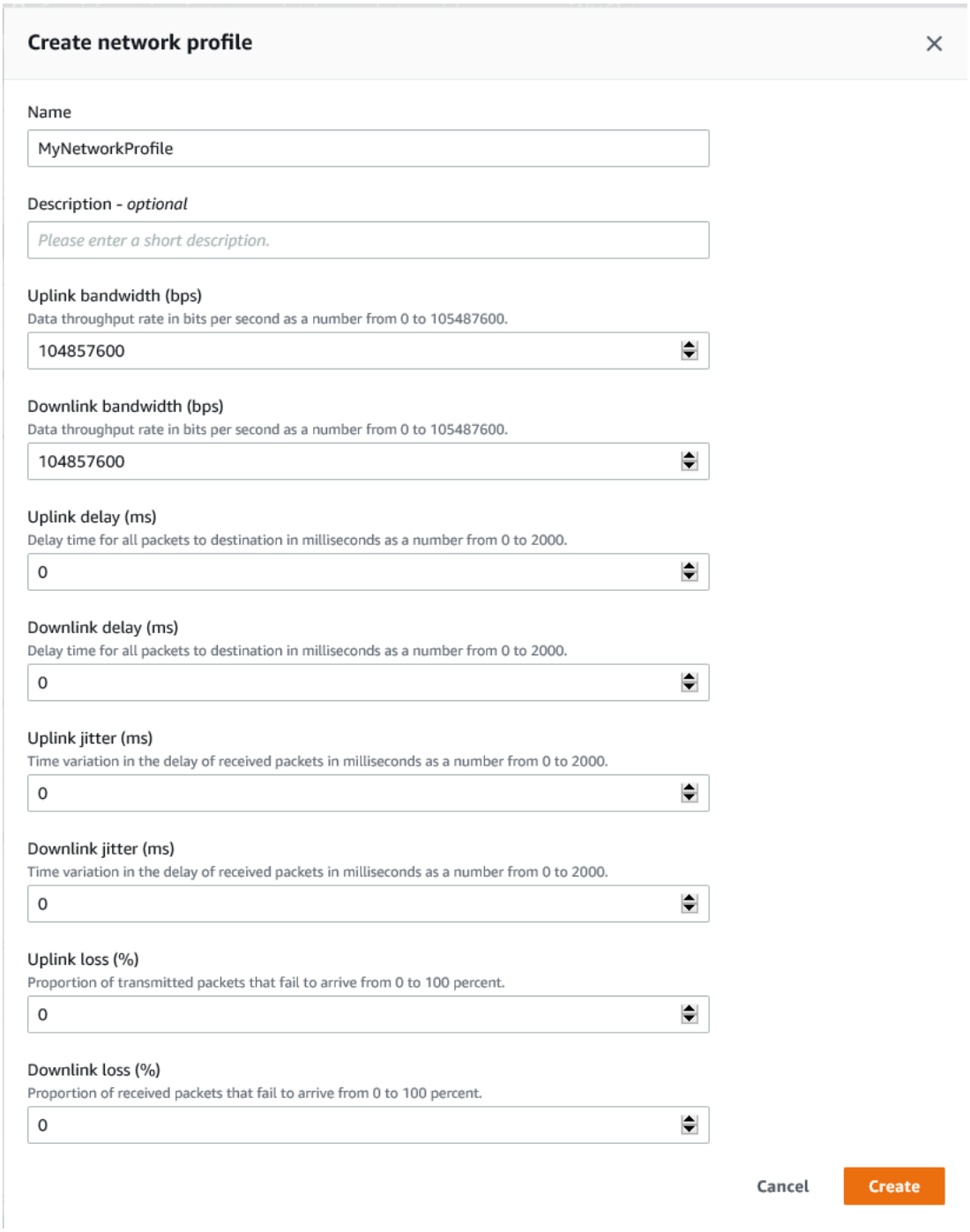



6. 다음을 선택하세요.
7. 테스트 실행을 검토하고 시작하세요.

## 네트워크 프로파일 생성

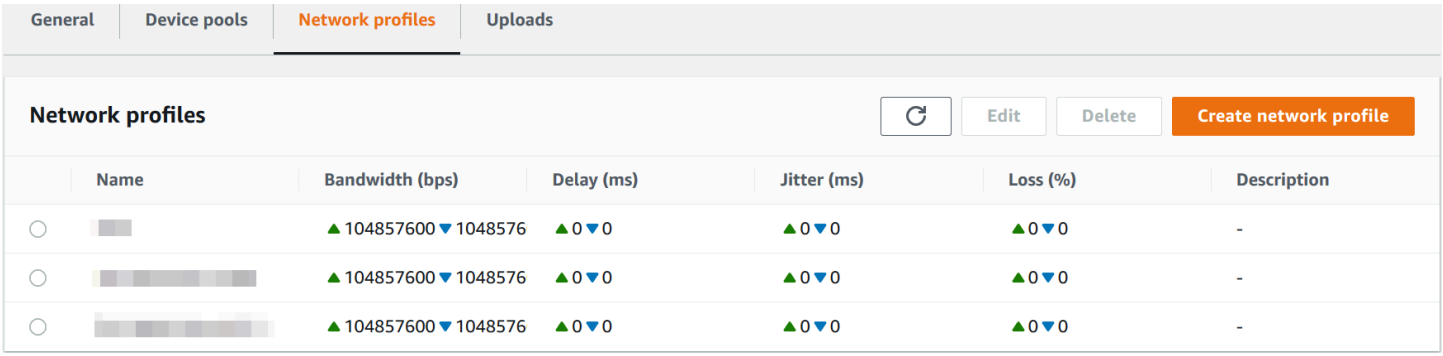
테스트 실행을 생성할 때 네트워크 프로필을 만들 수 있습니다.

1. 네트워크 프로필 생성을 선택하세요.




2. 네트워크 프로필의 이름과 설정을 입력하세요.
3. 생성을 선택하세요.
4. 테스트 실행 생성을 완료하고 실행을 시작하세요.

네트워크 프로필을 만든 후에는 프로젝트 설정 페이지에서 해당 프로필을 확인하고 관리할 수 있습니다.




## 테스트 중 네트워크 상태 변경

Appium 같은 프레임워크를 사용하여 디바이스 호스트에서 API를 직접 호출하여 테스트 실행 중 대역폭 감소와 같은 동적 네트워크 조건을 시뮬레이션할 수 있습니다. 자세한 내용은 CreateNetworkProfile를 참조하세요.

# AWS Device Farm에서의 실행 중지

시작한 후 실행을 중단하고 싶을 수도 있습니다. 예를 들어 테스트 실행 중에 문제가 발견되면 업데이트된 테스트 스크립트를 사용하여 실행을 다시 시작하는 것이 좋습니다.

Device Farm 콘솔 또는 API를 사용하여 실행을 중지할 수 있습니다. AWS CLI

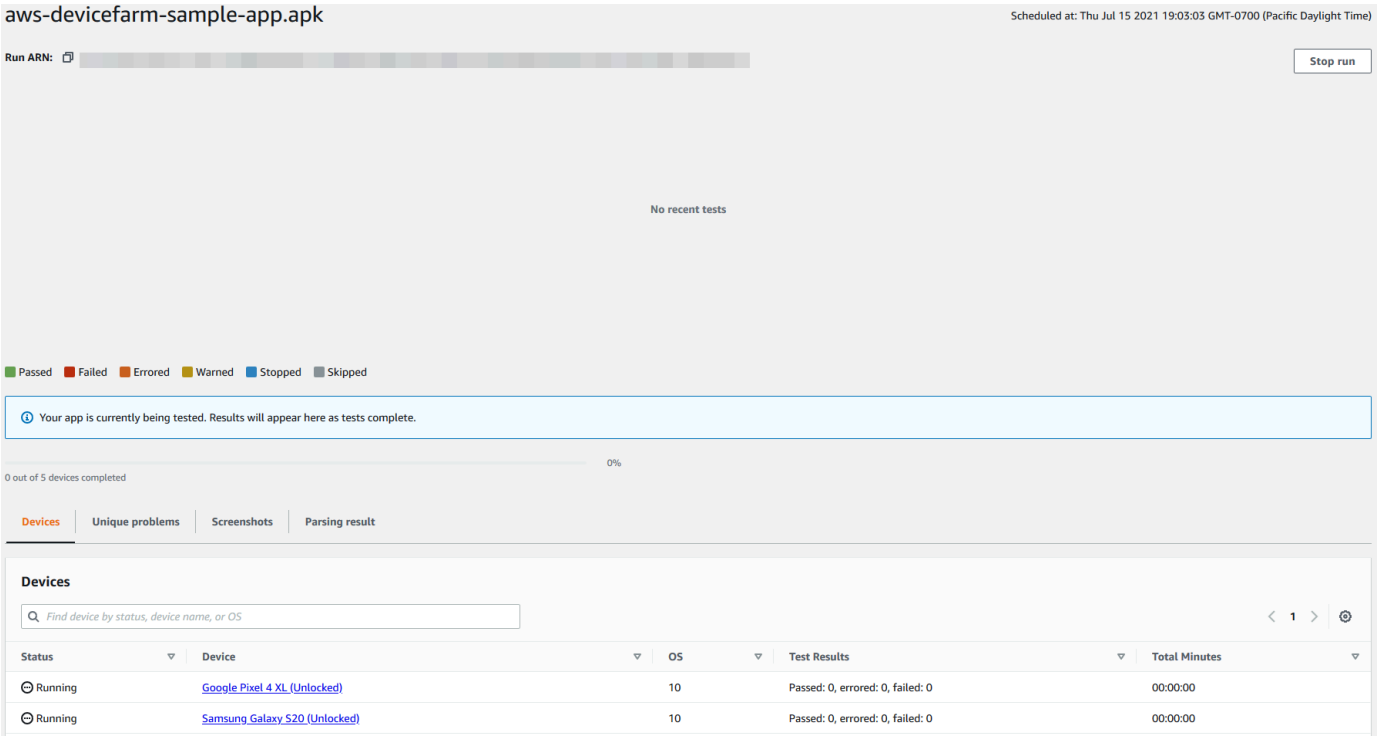
주제

- 실행 중지(콘솔)
- 실행 중지(AWS CLI)
- 실행 중지(API)

## 실행 중지(콘솔)

1. https://console.aws.amazon.com/devicefarm에서 Device Farm 콘솔에 로그인하세요.
2. Device Farm 탐색 패널에서 모바일 디바이스 테스트를 선택한 다음 프로젝트를 선택하세요.
3. 테스트 실행이 진행 중인 프로젝트를 선택하세요.
4. 자동 테스트 페이지에서 테스트 실행을 선택하세요.

   디바이스 이름 왼쪽에 보류 중 또는 실행 중 아이콘이 나타나야 합니다.

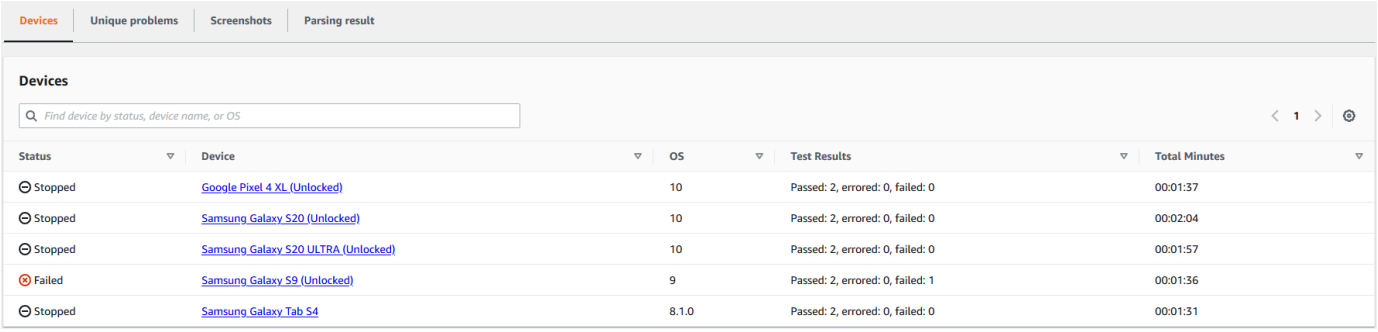



5. 실행 중지를 선택하세요.

잠시 후 디바이스 이름 옆에 빨간색 원 안에 마이너스가 있는 아이콘이 나타납니다. 실행이 중지되면 아이콘 색상이 빨간색에서 검은색으로 바뀝니다.

> **Important**
>
> 테스트가 이미 실행된 경우 Device Farm은 테스트를 중지할 수 없습니다. 테스트가 진행 중인 경우 Device Farm은 테스트를 중지합니다. 청구되는 총 시간(분)은 디바이스 섹션에 청구되어 표시됩니다. 또한 Device Farm에서 설치 스위트와 분해 스위트를 실행하는 데 걸리는 총 시간(분)에 대한 요금도 청구됩니다. 자세한 내용은 Device Farm 요금을 참조하세요.

다음 이미지는 테스트 실행이 성공적으로 중지된 후의 예제 디바이스 섹션을 보여줍니다.




# 실행 중지(AWS CLI)

다음 명령을 실행하여 지정된 테스트 실행을 중지할 수 있습니다. 여기서 *myARN*은 테스트 실행의 Amazon 리소스 이름(ARN)입니다.

```
$ aws devicefarm stop-run --arn myARN
```

다음과 유사한 출력 화면이 표시되어야 합니다.

```
{
    "run": {
        "status": "STOPPING",
        "name": "Name of your run",
        "created": 1458329687.951,
        "totalJobs": 7,
        "completedJobs": 5,
        "deviceMinutes": {
            "unmetered": 0.0,
            "total": 0.0,
            "metered": 0.0
        },
        "platform": "ANDROID_APP",
        "result": "PENDING",
        "billingMethod": "METERED",
        "type": "BUILTIN_EXPLORER",
        "arn": "myARN",
        "counters": {
            "skipped": 0,
            "warned": 0,
            "failed": 0,
            "stopped": 0,
            "passed": 0,
```

```
            "errored": 0,
            "total": 0
        }
    }
}
```

실행의 ARN을 가져오려면 `list-runs` 명령어를 사용합니다. 다음과 유사하게 출력됩니다.

```
{
    "runs": [
        {
            "status": "RUNNING",
            "name": "Name of your run",
            "created": 1458329687.951,
            "totalJobs": 7,
            "completedJobs": 5,
            "deviceMinutes": {
                "unmetered": 0.0,
                "total": 0.0,
                "metered": 0.0
            },
            "platform": "ANDROID_APP",
            "result": "PENDING",
            "billingMethod": "METERED",
            "type": "BUILTIN_EXPLORER",
            "arn": "Your ARN will be here",
            "counters": {
                "skipped": 0,
                "warned": 0,
                "failed": 0,
                "stopped": 0,
                "passed": 0,
                "errored": 0,
                "total": 0
            }
        }
    ]
}
```

Device Farm을 와 함께 사용하는 방법에 대한 자세한 내용은 을 AWS CLI참조하십시오AWS CLI 참조.

## 실행 중지(API)

- StopRun작업을 호출하여 테스트 실행하십시오.

Device Farm API 사용에 대한 자세한 내용은 Device Farm 자동화 단원을 참조하세요.

# AWS Device Farm에서 실행 목록 보기

Device Farm 콘솔 또는 API를 사용하여 프로젝트 실행 목록을 볼 수 있습니다. AWS CLI

주제

- 실행 목록 보기(콘솔)
- 실행 목록 보기(AWS CLI)
- 실행 목록 보기(API)

## 실행 목록 보기(콘솔)

1. https://console.aws.amazon.com/devicefarm에서 Device Farm 콘솔에 로그인하세요.
2. Device Farm 탐색 패널에서 모바일 디바이스 테스트를 선택한 다음 프로젝트를 선택하세요.
3. 프로젝트 목록에서 보려는 목록에 해당하는 프로젝트를 선택하세요.

> **Tip**
> 검색 창을 사용하여 프로젝트 목록을 이름별로 필터링할 수 있습니다.

## 실행 목록 보기(AWS CLI)

- list-runs 명령을 실행하세요.

  단일 실행에 대한 정보를 보려면 get-run 명령을 실행하세요.

Device Farm을 와 함께 사용하는 방법에 대한 자세한 내용은 을 AWS CLI참조하십시오AWS CLI 참조.

## 실행 목록 보기(API)

- `ListRuns` API를 호출하세요.

  단일 프로젝트에 대한 정보를 보려면 `GetRun` API를 호출하세요.

자세한 Device Farm API에 대한 내용은 Device Farm 자동화 단원을 참조하세요.

# AWS Device Farm에서 디바이스 풀 생성

Device Farm 콘솔 또는 API를 사용하여 장치 풀을 생성할 수 있습니다. AWS CLI

주제

- 사전 조건
- 디바이스 풀 생성(콘솔)
- 디바이스 풀(AWS CLI)을 생성하세요.
- 디바이스 풀(API) 생성

## 사전 조건

- Device Farm 콘솔에서 실행을 생성하세요. Device Farm에서 테스트 실행 생성의 지침을 따르세요. 디바이스 선택 페이지가 표시되면 현재 섹션의 지침을 계속 따르세요.

## 디바이스 풀 생성(콘솔)

1. 디바이스 선택 페이지에서 디바이스 풀 생성을 선택하세요.
2. 이름에는 이 디바이스 풀을 쉽게 식별할 수 있는 이름을 입력하세요.
3. 설명에는 이 디바이스 풀을 쉽게 식별할 수 있는 설명을 입력하세요.
4. 이 디바이스 풀의 디바이스에 대해 하나 이상의 선택 기준을 사용하려면 다음을 따르세요.

   a. 동적 디바이스 풀 생성을 선택하세요.

   b. 규칙 추가를 선택하세요.

   c. 필드(첫 번째 드롭다운 목록)의 경우 다음 중 하나를 선택하세요.

      - 제조업체 이름별로 디바이스를 포함하려면 디바이스 제조업체를 선택하세요.

- 유형 값을 기준으로 디바이스를 포함하려면 폼 팩터를 선택하세요.

d. 연산자(두 번째 드롭다운 목록)의 경우 같음을 선택하여 필드 값이 값 값과 같은 디바이스를 포함시키세요.

e. 값(세 번째 드롭다운 목록)에 필드 및 연산자 값에 지정할 값을 입력하거나 선택하세요. 필드용 플랫폼을 선택하면 ANDROID 및 IOS만 선택할 수 있습니다. 마찬가지로 필드용 폼 팩터를 선택하면 전화와 태블릿만 선택할 수 있습니다.

f. 다른 규칙을 추가하려면 규칙 추가를 선택하세요.

g. 규칙을 삭제하려면 규칙 옆의 X를 선택하세요.

첫 번째 규칙을 만들면 디바이스 목록에서 규칙과 일치하는 각 디바이스 옆의 상자가 선택됩니다. 추가로 규칙을 만들거나 변경하면 디바이스 목록에 조합된 규칙에 일치하는 각 디바이스 옆의 상자가 선택됩니다. 선택한 상자가 있는 디바이스는 디바이스 풀에 포함됩니다. 상자가 비워진 디바이스는 제외됩니다.

5. 개별 디바이스를 수동으로 포함하거나 제외하려면 다음을 수행하세요.

a. 정적 디바이스 풀 생성을 선택하세요.

b. 각 디바이스 옆의 상자를 선택하거나 지우세요. 규칙을 지정하지 않은 경우에만 상자를 선택하거나 선택을 취소할 수 있습니다.

6. 표시된 모든 디바이스를 포함하거나 제외하려면 목록의 열 헤더 행에 있는 상자를 선택하거나 선택 취소하세요.

> **Important**
>
> 열 헤더 행의 상자를 사용하여 표시된 디바이스 목록을 변경할 수 있지만 표시된 디바이스만 포함되거나 제외되는 것은 아닙니다. 어떤 디바이스가 포함되거나 제외되었는지 확인하려면 열 헤더 행에 있는 모든 상자의 내용을 지운 다음 상자를 둘러보세요.

7. 생성을 선택하세요

## 디바이스 풀(AWS CLI)을 생성하세요.

- create-device-pool 명령을 실행하세요.

Device Farm을 와 함께 사용하는 방법에 대한 자세한 내용은 을 AWS CLI참조하십시오AWS CLI 참조.

## 디바이스 풀(API) 생성

- `CreateDevicePool` API를 호출하세요.

자세한 Device Farm API 사용 방법은 Device Farm 자동화 단원을 참조하세요.

# AWS Device Farm에서 결과 분석

표준 테스트 환경에서 Device Farm 콘솔을 사용하여 테스트 실행의 각 테스트에 대한 보고서를 볼 수 있습니다.

또한 Device Farm은 테스트 실행 완료 시 다운로드할 수 있는 파일, 로그, 이미지와 같은 기타 아티팩트를 수집합니다.

주제

- Device Farm에서 테스트 보고서 작업
- Device Farm에서 아티팩트 작업

## Device Farm에서 테스트 보고서 작업

Device Farm 콘솔을 사용하여 테스트 보고서를 볼 수 있습니다. 자세한 내용은 AWS Device Farm에 있는 보고서 단원을 참조하세요.

주제

- 사전 조건
- 테스트 결과의 이해
- 보고서 보기

### 사전 조건

테스트 실행을 설정하고 완료되었는지 확인하세요.

1. 실행을 생성하려면 Device Farm에서 테스트 실행 생성을 참조한 다음 이 페이지로 돌아오세요.
2. 실행이 완료되었는지 확인 테스트 실행 중에 Device Farm 콘솔에는 진행 중인 실행에 대한 보류 중 아이콘


이

표시됩니다. 실행 중인 각 디바이스도 보류 중 아이콘으로 시작하다가 테스트가 시작되면 실행 중 아이콘


로

전환됩니다. 각 테스트가 완료되면 디바이스 이름 옆에 테스트 결과 아이콘이 표시됩니다. 모든 테스트가 완료되면 실행 옆에 있는 보류 중 아이콘이 테스트 결과 아이콘으로 바뀝니다. 자세한 내용은 테스트 결과의 이해 단원을 참조하세요.

## 테스트 결과의 이해

Device Farm 콘솔에는 완료된 테스트 실행의 상태를 빠르게 평가하는 데 도움이 되는 아이콘이 표시됩니다.

주제

- 개별 테스트에 대한 결과 보고
- 여러 테스트에 대한 결과 보고

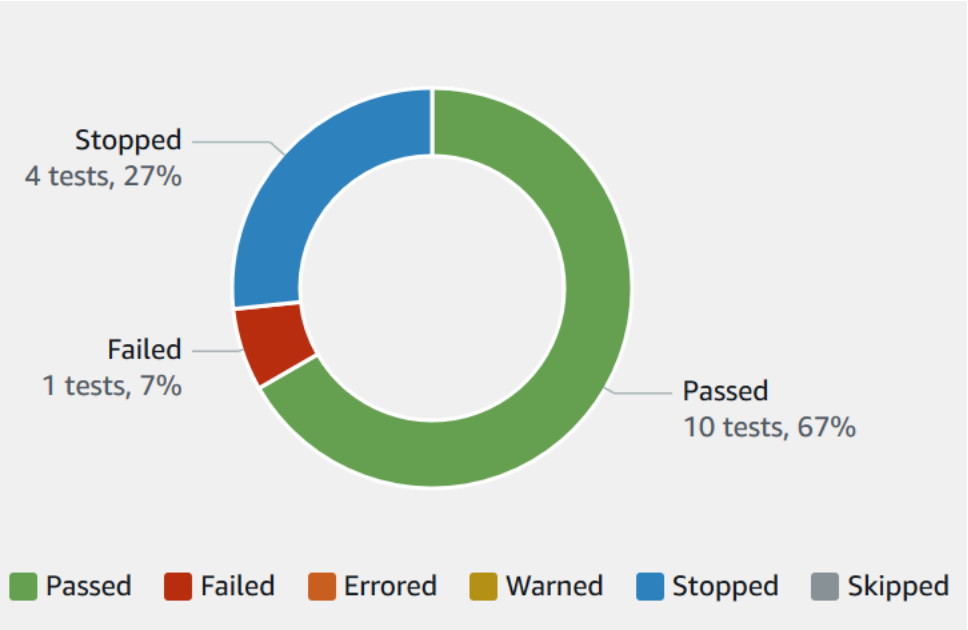
### 개별 테스트에 대한 결과 보고

개별 테스트를 설명하는 보고서의 경우 Device Farm이 다음 아이콘을 표시합니다.

| 설명 | 아이콘 |
| --- | --- |
| 테스트가 성공했습니다. |  |
| 테스트에 실패했습니다. |  |
| Device Farm은 테스트를 건너뛰었습니다. |  |
| 테스트가 중단되었습니다. |  |
| Device Farm에서 경고를 반환했습니다. |  |
| Device Farm에서 오류가 반환되었습니다. |  |

여러 테스트에 대한 결과 보고

실행을 완료한 경우 Device Farm은 테스트 결과 요약 그래프를 표시합니다.




예를 들어 이 테스트 실행 결과 그래프는 중지된 테스트 4개, 실패한 테스트 1개, 성공한 테스트 10개를 보여줍니다.

그래프는 항상 색으로 구분되고 레이블이 지정됩니다.

## 보고서 보기

Device Farm 콘솔에서 테스트 결과를 볼 수 있습니다.

주제

- 테스트 실행 요약 페이지 보기
- 고유한 문제 보고서 보기
- 디바이스 보고서 보기
- 테스트 스위트 보고서 보기
- 테스트 보고서 보기
- 보고서에서 문제, 디바이스, 스위트 및 테스트 성능 데이터 보기
- 보고서에서 문제, 디바이스, 스위트 및 테스트 로그 정보 보기

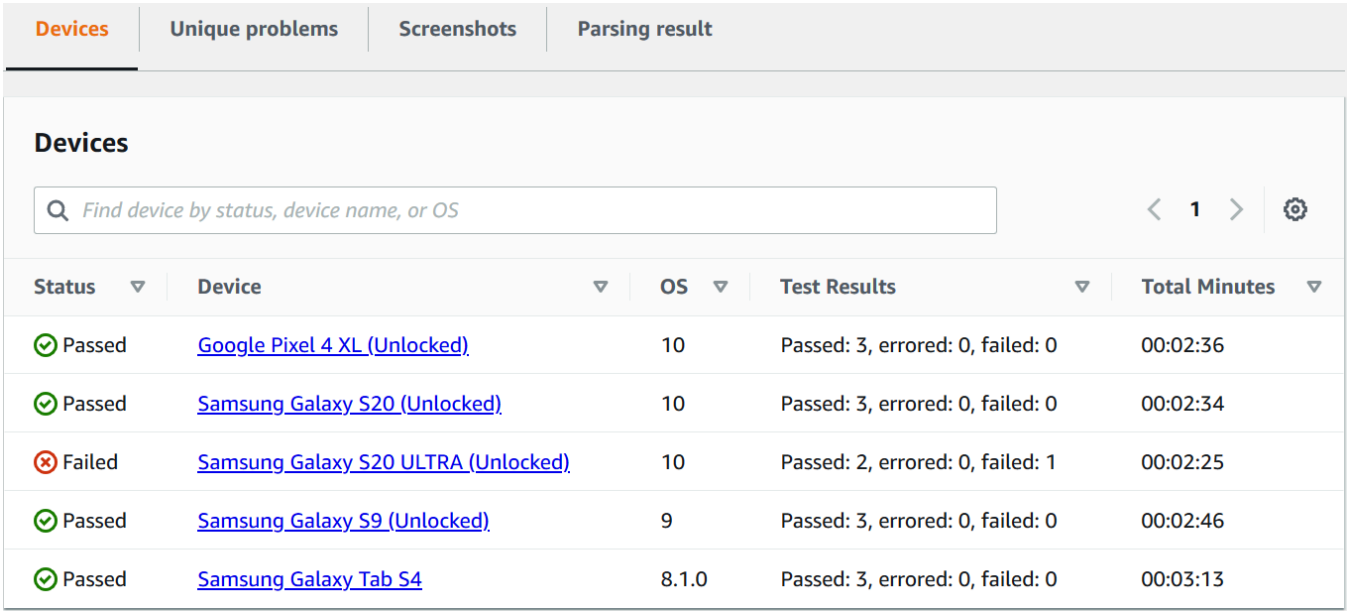
테스트 실행 요약 페이지 보기

1. https://console.aws.amazon.com/devicefarm에서 Device Farm 콘솔에 로그인하세요.
2. 탐색 창에서 모바일 디바이스 테스트를 선택하고 프로젝트를 선택하세요.
3. 프로젝트 목록에서 실행할 프로젝트를 선택하세요.

> **Tip**
> 검색 창을 사용하여 리소스 이름별로 목록을 필터링하세요.

4. 완료된 실행을 선택하면 요약 보고서 페이지를 볼 수 있습니다.
5. 테스트 실행 요약 페이지에는 테스트 결과 개요가 표시됩니다.

   - 고유한 문제 섹션에는 고유한 경고 및 실패가 나열되어 있습니다. 고유한 문제를 보려면 고유한 문제 보고서 보기의 지침을 따르세요.
   - 디바이스 섹션에는 각 디바이스에 대한 총 테스트 수가 결과별로 표시됩니다.




   이 예시는 여러 디바이스를 포함합니다. A첫 번째 표 항목에서 Android 버전 10을 실행하는 Google Pixel 4 XL 디바이스는 실행에 02:36분 소요된 3개 테스트를 성공적으로 완료했다고 보고합니다.

   디바이스별로 결과를 보려면 디바이스 보고서 보기 단원의 안내를 따르세요.

- 스크린샷 섹션에는 Device Farm이 실행 중에 캡처한 스크린샷 목록이 디바이스별로 그룹화되어 표시됩니다.
- 파싱 결과 섹션에서 파싱 결과를 다운로드할 수 있습니다.

### 고유한 문제 보고서 보기

1. 고유한 문제에서 보려는 문제를 선택하세요.
2. 디바이스를 선택하세요. 보고서에는 문제에 대한 정보가 표시됩니다.

   비디오 섹션에는 테스트의 다운로드 가능한 비디오 녹화가 표시됩니다.

   결과 섹션에는 테스트 결과가 표시됩니다. 상태는 결과 아이콘으로 표시됩니다. 자세한 내용은 개별 테스트에 대한 결과 보고 단원을 참조하세요.

   로그 섹션에는 Device Farm이 테스트 중에 기록한 모든 정보가 표시됩니다. 이 정보를 보려면 보고서에서 문제, 디바이스, 스위트 및 테스트 로그 정보 보기의 지침을 따르세요.

   성능 탭에는 Device Farm이 테스트 중에 생성한 모든 성능 데이터에 대한 정보가 표시됩니다. 성능 데이터를 보려면 보고서에서 문제, 디바이스, 스위트 및 테스트 성능 데이터 보기의 지침을 따르세요.

   파일 탭에는 다운로드할 수 있는 모든 테스트 관련 파일(예: 로그 파일)의 목록이 표시됩니다. 파일을 다운로드하려면 목록에서 파일 링크를 선택하세요.

   스크린샷 탭에는 Device Farm이 테스트 중에 캡처한 스크린샷 목록이 표시됩니다.

### 디바이스 보고서 보기

- 디바이스 섹션에서 디바이스를 선택하세요.

  비디오 섹션에는 테스트의 다운로드 가능한 비디오 녹화가 표시됩니다.

  스위트 섹션에는 디바이스의 스위트 정보가 담긴 표가 표시됩니다.

  이 표의 테스트 결과 열은 디바이스에서 실행된 각 테스트 스위트의 테스트 수가 결과별로 요약되어 있습니다. 이 데이터에는 그래픽 구성 요소 또한 포함합니다. 자세한 내용은 여러 테스트에 대한 결과 보고 단원을 참조하세요.

  스위트별 전체 결과를 보려면 테스트 스위트 보고서 보기 단원의 지침을 따르세요.

로그 섹션에는 Device Farm이 실행 중에 디바이스에 대해 기록한 모든 정보가 표시됩니다. 이 정보를 보려면 보고서에서 문제, 디바이스, 스위트 및 테스트 로그 정보 보기의 지침을 따르세요.

성능 섹션에는 Device Farm이 실행 중에 디바이스에 대해 생성한 모든 성능 데이터에 대한 정보가 표시됩니다. 성능 데이터를 보려면 보고서에서 문제, 디바이스, 스위트 및 테스트 성능 데이터 보기의 지침을 따르세요.

파일 섹션에는 디바이스의 스위트 목록과 다운로드할 수 있는 관련 파일(예: 로그 파일)이 표시됩니다. 파일을 다운로드하려면 목록에서 파일 링크를 선택하세요.

스크린샷 섹션에는 Device Farm이 디바이스 실행 중에 캡처한 스크린샷 목록이 세트별로 그룹화되어 표시됩니다.

### 테스트 스위트 보고서 보기

1. 디바이스 섹션에서 디바이스를 추가하세요.
2. 스위트 섹션의 표에 있는 스위트를 선택하세요.

   비디오 섹션에는 테스트의 다운로드 가능한 비디오 녹화가 표시됩니다.

   테스트 섹션에는 스위트의 테스트 정보가 담긴 표가 표시됩니다.

   표의 테스트 결과 열에는 결과가 표시됩니다. 이 데이터에는 그래픽 구성 요소도 또한 포함합니다. 자세한 내용은 여러 테스트에 대한 결과 보고 단원을 참조하세요.

   테스트별 전체 결과를 보려면 테스트 보고서 보기의 지침을 따르세요.

   로그 섹션에는 스위트 실행 중에 Device Farm이 기록한 모든 정보가 표시됩니다. 이 정보를 보려면 보고서에서 문제, 디바이스, 스위트 및 테스트 로그 정보 보기의 지침을 따르세요.

   성능 섹션에는 스위트 실행 중에 Device Farm이 생성한 모든 성능 데이터에 대한 정보가 표시됩니다. 성능 데이터를 보려면 보고서에서 문제, 디바이스, 스위트 및 테스트 성능 데이터 보기의 지침을 따르세요.

   파일 섹션에는 스위트의 테스트 목록과 다운로드할 수 있는 관련 파일(예: 로그 파일)이 표시됩니다. 파일을 다운로드하려면 목록에서 파일 링크를 선택하세요.

   스크린샷 섹션에는 스위트 실행 중에 Device Farm이 캡처한 스크린샷 목록이 테스트별로 그룹화되어 표시됩니다.

테스트 보고서 보기

1. 디바이스 섹션에서 디바이스를 선택하세요.
2. 스위트 섹션에서 스위트를 선택하세요.
3. 테스트 섹션에서 테스트를 선택하세요.
4. 비디오 섹션에는 테스트의 다운로드 가능한 비디오 녹화가 표시됩니다.

   결과 섹션에는 테스트 결과가 표시됩니다. 상태는 결과 아이콘으로 표시됩니다. 자세한 내용은 개별 테스트에 대한 결과 보고 단원을 참조하세요.

   로그 섹션에는 Device Farm이 테스트 중에 기록한 모든 정보가 표시됩니다. 이 정보를 보려면 보고서에서 문제, 디바이스, 스위트 및 테스트 로그 정보 보기의 지침을 따르세요.

   성능 탭에는 Device Farm이 테스트 중에 생성한 모든 성능 데이터에 대한 정보가 표시됩니다. 성능 데이터를 보려면 보고서에서 문제, 디바이스, 스위트 및 테스트 성능 데이터 보기의 지침을 따르세요.

   파일 탭에는 다운로드할 수 있는 모든 테스트 관련 파일(예: 로그 파일)의 목록이 표시됩니다. 파일을 다운로드하려면 목록에서 파일 링크를 선택하세요.

   스크린샷 탭에는 Device Farm이 테스트 중에 캡처한 스크린샷 목록이 표시됩니다.

보고서에서 문제, 디바이스, 스위트 및 테스트 성능 데이터 보기

> **Note**
>
> Device Farm은 현재 Android 디바이스에 대한 디바이스 성능 데이터만 수집합니다.

성능 탭에는 다음 정보가 표시됩니다.

- CPU 그래프는 선택한 문제, 디바이스, 스위트 또는 테스트(세로축 기준)에 대하여 앱이 단일 코어에서 사용한 CPU의 시간(가로축 기준)에 따른 비율을 표시합니다.

  세로축은 0%에서 기록된 최대 백분율까지의 백분율로 표시됩니다.

  앱에서 코어를 두 개 이상 사용한 경우 이 비율은 100%를 초과할 수 있습니다. 예를 들어 3개 코어의 사용량이 60%인 경우 이 비율은 180%로 표시됩니다.

- 메모리 그래프에는 선택한 문제, 디바이스, 스위트 또는 테스트(세로축 기준) 중에 앱이 사용한 MB 수가 시간 경과(가로축 기준)에 따라 표시됩니다.

  세로축은 0MB부터 기록된 최대 MB 수까지 MB 단위로 표시됩니다.
- 스레드 그래프는 선택한 문제, 디바이스, 스위트 또는 테스트(세로축 기준) 중에 사용된 스레드 수를 시간 경과(가로축 기준)에 따라 표시합니다.

  세로축은 스레드 수로 표시되며, 스레드 수는 0개부터 기록된 최대 스레드 수까지입니다.

모든 경우에 가로축은 선택한 문제, 디바이스, 스위트 또는 테스트의 실행 시작 및 종료 시점부터 초 단위로 표시됩니다.

특정 데이터 포인트에 대한 정보를 표시하려면 원하는 그래프에서 가로축을 따라 원하는 초 단위에서 잠시 멈추세요.

보고서에서 문제, 디바이스, 스위트 및 테스트 로그 정보 보기

로그 섹션에는 다음 정보가 표시됩니다.

- 소스는 로그 항목의 출처를 나타냅니다. 가능한 값은 다음과 같습니다.
  - 도구는 Device Farm에서 생성한 로그 항목을 나타냅니다. 이러한 로그 항목은 일반적으로 시작 및 중지 이벤트 중에 생성됩니다.
  - 디바이스는 디바이스에서 생성한 로그 항목을 나타냅니다. Android의 경우 이러한 로그 항목은 logcat과 호환됩니다. iOS의 경우 이러한 로그 항목은 syslog와 호환됩니다.
  - 테스트는 테스트 또는 테스트 프레임워크에서 만든 로그 항목을 나타냅니다.
- 시간은 첫 번째 로그 항목과 이 로그 항목 사이의 경과 시간을 나타냅니다. 시간은 *MM:SS.SSS* 형식으로 표시되며, 여기서 *M*은 분을 나타내고 *S*는 초를 나타냅니다.
- PID는 로그 항목을 생성한 프로세스 식별자(PID)를 나타냅니다. 디바이스에서 앱으로 만든 모든 로그 항목은 동일한 PID를 가집니다.
- 레벨은 로그 항목의 로깅 레벨을 나타냅니다. 예를 들어, `Logger.debug("This is a message!")`는 Debug의 레벨를 로깅합니다. 사용 가능한 값은 다음과 같습니다.
  - 알림
  - 위험
  - 디버그
  - 긴급 상황
  - 오류

- 오류 발생
- 실패
- 정보
- 내부
- 알림
- 통과
- 건너뜀
- 중지됨
- 상세 표시
- 경고
- 경고

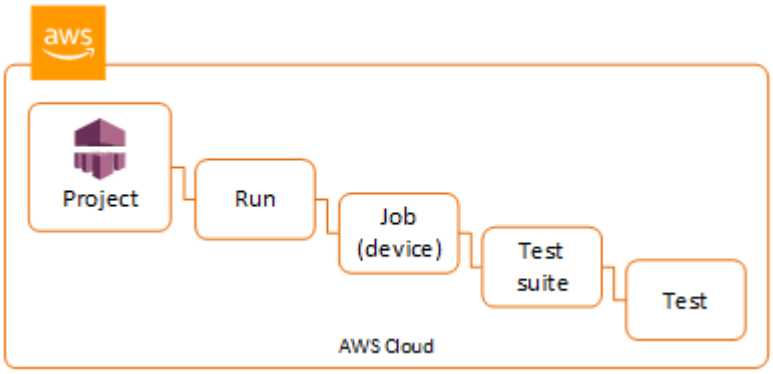
- 태그는 로그 항목의 임의 메타데이터를 나타냅니다. 예를 들어 Android logcat은 이를 사용하여 시스템의 어느 부분에서 로그 항목을 생성했는지 설명할 수 있습니다(예: `ActivityManager`).
- 메시지는 로그 항목에 대한 메시지 또는 데이터를 나타냅니다. 예를 들어, `Logger.debug("Hello, World!")`은 `"Hello, World!"`의 메시지를 로깅합니다.

정보의 일부만 표시하려면 다음을 따르세요.

- 특정 열의 값과 일치하는 모든 로그 항목을 표시하려면 검색 창에 값을 입력합니다. 예를 들어, 소스 값이 `Harness`인 모든 로그 항목을 표시하려면 검색 창에 **`Harness`**를 입력합니다.
- 열 헤더 상자에서 모든 문자를 제거하려면 해당 열 헤더 상자에서 X를 선택하세요. 열 헤더 상자에서 모든 문자를 제거하는 것은 해당 열 헤더 상자에 **`*`**을 입력하는 것과 같습니다.

실행한 모든 스위트 및 테스트를 포함하는 디바이스에 대한 모든 로그 정보를 다운로드하려면 로그 다운로드를 선택하세요.

# Device Farm에서 아티팩트 작업




Device Farm은 실행 중인 각 테스트에 대한 보고서, 로그 파일 및 이미지와 같은 아티팩트를 수집합니다.

테스트 실행 중에 생성된 아티팩트를 다운로드할 수 있습니다.

파일

Device Farm 보고서를 포함하여 테스트 실행 중에 생성된 파일입니다. 자세한 내용은 Device Farm에서 테스트 보고서 작업 단원을 참조하세요.

로그

테스트 실행의 각 테스트 출력

스크린샷

테스트 실행 시 각 테스트가 기록된 화면 이미지

## 아티팩트 사용(콘솔)

1. 테스트 실행 보고서 페이지의 디바이스에서 모바일 디바이스를 선택하세요.
2. 파일을 다운로드하려면 파일에서 선택하세요.
3. 테스트 실행에서 로그를 다운로드하려면 로그에서 로그 다운로드를 선택하세요.
4. 스크린샷을 다운로드하려면 스크린샷에서 스크린샷을 선택하세요.

사용자 지정 테스트 환경에 대한 자세한 내용은 사용자 지정 테스트 환경에서 아티팩트 사용 단원을 참조하세요.

## 아티팩트 사용(AWS CLI)

를 사용하여 테스트 실행 아티팩트를 AWS CLI 나열할 수 있습니다.

주제

- 1단계: Amazon 리소스 이름(ARN) 가져오기
- 2단계: 아티팩트 목록 작성
- 3단계: 아티팩트 다운로드

### 1단계: Amazon 리소스 이름(ARN) 가져오기

실행, 작업, 테스트 스위트 또는 테스트별로 아티팩트를 나열할 수 있습니다. 해당하는 ARN이 필요합니다. 다음 표는 각 AWS CLI 목록 명령에 대한 입력 ARN을 보여줍니다.

| AWS CLI 목록 명령 | 필수 ARN |
| --- | --- |
| list-projects | 이 명령은 모든 프로젝트를 반환하며 ARN이 필요하지 않습니다. |
| list-runs | `project` |
| list-jobs | `run` |
| list-suites | `job` |
| list-tests | `suite` |

예를 들어 테스트 ARN을 찾으려면 테스트 스위트 ARN을 입력 매개변수로 사용하여 list-tests을 실행하세요.

예제

```
aws devicefarm list-tests --arn arn:MyTestSuiteARN
```

응답에는 테스트 스위트의 각 테스트에 대한 테스트 ARN이 포함됩니다.

```
{
    "tests": [
```

```
        {
            "status": "COMPLETED",
            "name": "Tests.FixturesTest.testExample",
            "created": 1537563725.116,
            "deviceMinutes": {
                "unmetered": 0.0,
                "total": 1.89,
                "metered": 1.89
            },
            "result": "PASSED",
            "message": "testExample passed",
            "arn": "arn:aws:devicefarm:us-west-2:123456789101:test:5e01a8c7-c861-4c0a-
b1d5-12345EXAMPLE",
            "counters": {
                "skipped": 0,
                "warned": 0,
                "failed": 0,
                "stopped": 0,
                "passed": 1,
                "errored": 0,
                "total": 1
            }
        }
    ]
}
```

2단계: 아티팩트 목록 작성

AWS CLI list-artifacts 명령은 파일, 스크린샷, 로그와 같은 아티팩트 목록을 반환합니다. 각 아티팩트에는 파일을 다운로드할 수 있는 URL이 있습니다.

- 실행, 작업, 테스트 스위트 또는 테스트 ARN을 지정하여 list-artifacts를 호출합니다. 파일, 로그 또는 스크린샷 유형을 지정합니다.

  이 예제는 개별 테스트에 사용할 수 있는 각 아티팩트의 다운로드 URL을 반환합니다.

  ```
  aws devicefarm list-artifacts --arn arn:MyTestARN --type "FILE"
  ```

  응답에는 각 아티팩트의 다운로드 URL이 포함됩니다.

  ```
  {
      "artifacts": [
  ```

```
        {
            "url": "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/
ExampleURL",
            "extension": "txt",
            "type": "APPIUM_JAVA_OUTPUT",
            "name": "Appium Java Output",
            "arn": "arn:aws:devicefarm:us-west-2:123456789101:artifact:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE",
        }
    ]
}
```

3단계: 아티팩트 다운로드

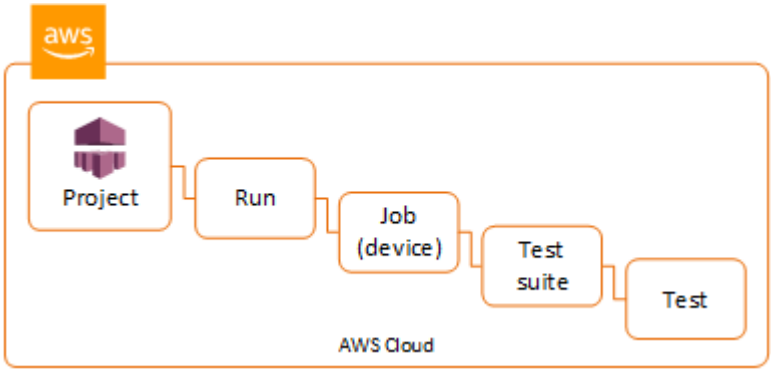
- 이전 단계의 URL을 사용하여 아티팩트를 다운로드합니다. 이 예제에서는 Android Appium Java 출력 파일을 다운로드하는 데 curl을 사용합니다.

```
curl "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/ExampleURL"
 > MyArtifactName.txt
```

## 아티팩트(API) 사용

Device Farm API ListArtifacts메서드는 파일, 스크린샷, 로그와 같은 아티팩트 목록을 반환합니다. 각 아티팩트에는 파일을 다운로드할 수 있는 URL이 있습니다.

## 사용자 지정 테스트 환경에서 아티팩트 사용




사용자 지정 테스트 환경에서 Device Farm은 사용자 지정 보고서, 로그 파일 및 이미지와 같은 아티팩트를 수집합니다. 이러한 테스트 아티팩트는 테스트 실행에서 각 디바이스에 사용할 수 있습니다.

테스트 실행 중에 생성된 다음 아티팩트를 다운로드할 수 있습니다.

테스트 사양 출력

테스트 사양 YAML 파일에서 명령을 실행한 결과입니다.

고객 아티팩트

테스트 실행의 아티팩트가 포함된 압축 파일입니다. 테스트 사양 YAML 파일의 artifacts: 섹션에서 구성됩니다.

테스트 사양 쉘 스크립트

YAML 파일에서 만든 중급 쉘 스크립트 파일입니다. 쉘 스크립트 파일은 테스트 실행에 사용되므로 YAML 파일을 디버깅하는 데 사용할 수 있습니다.

테스트 사양 파일

테스트 실행에 사용된 YAML 파일입니다.

자세한 내용은 Device Farm에서 아티팩트 작업 단원을 참조하세요.

# AWS Device Farm 리소스에 태그 지정

AWS Device Farm은 AWS 리소스 그룹 태그 지정 API와 함께 작동합니다. 이 API를 사용하면 태그로 AWS 계정의 리소스를 관리할 수 있습니다. 프로젝트 및 테스트 실행과 같은 리소스에 태그를 추가할 수 있습니다.

태그를 사용하여 다음을 수행할 수 있습니다.

- 비용 구조를 반영하도록 AWS 대금을 구성합니다. 이를 위해 가입하여 태그 키 값이 포함된 AWS 계정 청구서를 가져옵니다. 그런 다음 같은 태그 키 값을 가진 리소스에 따라 결제 정보를 구성하여 리소스 비용의 합을 볼 수 있습니다. 예를 들어 애플리케이션 이름으로 여러 리소스에 태그를 지정한 다음, 결제 정보를 구성하여 여러 서비스에 걸친 해당 애플리케이션의 총 비용을 볼 수 있습니다. 자세한 내용은 AWS 과금 정보 및 비용 관리 설명서의 비용 할당과 태그 지정을 참조하세요.
- IAM 정책을 통해 액세스를 제어합니다. 이렇게 하려면 태그 값 조건을 사용하여 리소스 또는 리소스 집합에 대한 액세스를 허용하는 정책을 생성합니다.
- 테스트에 사용되는 브랜치와 같은 특정 속성을 태그로 가진 실행을 식별하고 관리합니다.

리소스 태그 지정에 대한 자세한 내용은 태그 지정 모범 사례 백서를 참조하세요.

토픽

- 리소스에 태그 지정
- 태그로 리소스 조회
- 리소스에서 태그 제거

## 리소스에 태그 지정

AWS Resource Group Tagging API를 사용하면 리소스에 대한 태그를 추가, 제거 또는 수정할 수 있습니다. 자세한 내용은 AWS Resource Groups 태그 지정 API 참조를 참조하세요.

리소스에 태그를 지정하려면 `resourcegroupstaggingapi` 엔드포인트에서 `TagResources` 작업을 사용하세요. 이 작업은 지원되는 서비스의 ARN 목록과 키 값 페어 목록을 가져옵니다. 값은 선택 사항입니다. 빈 문자열은 해당 태그의 값이 없어야 함을 나타냅니다. 예를 들어, 다음 Python 예제는 일련의 프로젝트 ARN에 `release` 값이 있는 `build-config` 태그를 지정합니다.

```
import boto3
```

```
client = boto3.client('resourcegroupstaggingapi')

client.tag_resources(ResourceARNList=["arn:aws:devicefarm:us-
west-2:111122223333:project:123e4567-e89b-12d3-a456-426655440000",
                                      "arn:aws:devicefarm:us-
west-2:111122223333:project:123e4567-e89b-12d3-a456-426655441111",
                                      "arn:aws:devicefarm:us-
west-2:111122223333:project:123e4567-e89b-12d3-a456-426655442222"]
                     Tags={"build-config":"release", "git-commit":"8fe28cb"})
```

태그 값은 필요하지 않습니다. 값 없이 태그를 설정하려면 값을 지정할 때 빈 문자열("")을 사용하세요. 태그는 하나의 값만 가질 수 있습니다. 리소스에 대한 태그의 이전 값은 새 값으로 덮어쓰게 됩니다.

## 태그로 리소스 조회

태그로 리소스를 조회하려면 `resourcegrouptaggingapi` 엔드포인트의 `GetResources` 작업을 사용하세요. 이 작업은 일련의 필터(필수가 아님)를 사용하여 지정된 기준과 일치하는 리소스를 반환합니다. 필터가 없으면 태그가 지정된 모든 리소스가 반환됩니다. `GetResources` 작업을 통해 다음을 기준으로 리소스를 필터링할 수 있습니다.

- 태그 값
- 리소스 유형(예: `devicefarm:run`)

자세한 내용은 AWS Resource Group 태그 지정 API 참조를 참조하세요.

다음 예제에서는 `production` 값이 있는 `stack` 태그로 Device Farm 데스크톱 브라우저 테스트 세션(`devicefarm:testgrid-session` 리소스)을 조회합니다.

```
import boto3
client = boto3.client('resourcegroupstaggingapi')
sessions = client.get_resources(ResourceTypeFilters=['devicefarm:testgrid-session'],
                                TagFilters=[
                                  {"Key":"stack","Values":["production"]}
                                ])
```

## 리소스에서 태그 제거

태그를 제거하려면 제거할 리소스 및 태그 목록을 지정하여 `UntagResources` 작업을 사용하세요.

```
import boto3
client = boto3.client('resourcegroupstaggingapi')
client.UntagResources(ResourceARNList=["arn:aws:devicefarm:us-
west-2:111122223333:project:123e4567-e89b-12d3-a456-426655440000"], TagKeys=["RunCI"])
```

# AWS Device Farm에서 테스트 유형을 사용한 작업

이 섹션에서는 테스트 프레임워크 및 기본 제공 테스트 유형에 대한 Device Farm 지원을 설명합니다.

## 테스트 프레임워크

Device Farm은 다음과 같은 모바일 자동화 테스트 프레임워크를 지원합니다.

### Android 애플리케이션 테스트 프레임워크

- Appium 및 AWS Device Farm과 함께 작업
- Android 및 AWS Device Farm용 계측 작업

### iOS 애플리케이션 테스트 프레임워크

- Appium 및 AWS Device Farm과 함께 작업
- iOS용 XCTest 및 AWS Device Farm과 함께 작업
- XCTest UI

### 웹 애플리케이션 테스트 프레임워크

웹 애플리케이션은 Appium을 사용하여 지원됩니다. 테스트를 Appium으로 가져오는 방법에 대한 자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

### 사용자 지정 테스트 환경의 프레임워크

Device Farm은 XCTests 프레임워크를 위한 테스트 환경의 사용자 지정을 지원하지 않습니다. 자세한 정보는 사용자 지정 테스트 환경 작업을 참조하세요.

### Appium 버전 지원

사용자 지정 환경에서 실행되는 테스트의 경우 Device Farm은 Appium 버전 1을 지원합니다. 자세한 정보는 테스트 환경을 참조하세요.

# 기본 제공 테스트 유형

기본 제공 테스트의 경우 테스트 자동화 스크립트를 작성 및 유지 관리하지 않고도 여러 디바이스에서 애플리케이션을 테스트할 수 있습니다. Device Farm은 다음의 두 가지 기본 제공 테스트 유형을 제공합니다. Device Farm은 다음과 같은 한 가지 기본 테스트 유형을 제공합니다.

- 내장: fuzz (Android 및 iOS)

# Appium 및 AWS Device Farm과 함께 작업

이 섹션에서는 Appium 테스트를 구성 및 패키징하고 Device Farm에 업로드하는 방법을 설명합니다. Appium은 네이티브 애플리케이션 및 모바일 웹 애플리케이션을 자동화하는 오픈 소스 도구입니다. 자세한 정보는 Appium 웹 사이트에서 Appium 소개를 참조하세요.

샘플 앱과 작동 중인 테스트에 대한 링크는 Android용 Device Farm 샘플 앱 및 iOS용 Device Farm 샘플 앱 온을 참조하십시오 GitHub.

## 버전 지원

다양한 프레임워크 및 프로그래밍 언어에 대한 지원은 사용되는 언어에 따라 다릅니다.

Device Farm은 Appium 1.x 및 2.x 서버 버전을 지원합니다. Android의 경우 `devicefarm-cli`로 Appium 주요 버전을 선택하여 사용할 수 있습니다. 예를 들어 Appium 서버 버전 2를 사용하려면 다음 명령을 테스트 사양 YAML 파일에 추가합니다.

```
phases:
  install:
    commands:
      # To install a newer version of Appium such as version 2:
      - export APPIUM_VERSION=2
      - devicefarm-cli use appium $APPIUM_VERSION
```

iOS의 경우 `avm` 또는 `npm` 명령을 사용하여 특정 Appium 버전을 선택할 수 있습니다. 예를 들어 Appium 서버 버전을 2.1.2로 설정하기 위해 `avm` 명령을 사용하려면 다음 명령을 테스트 사양 YAML 파일에 추가하세요.

```
phases:
  install:
```

```
    commands:
      # To install a newer version of Appium such as version 2.1.2:
      - export APPIUM_VERSION=2.1.2
      - avm $APPIUM_VERSION
```

최신 버전의 Appium 2를 사용하는 npm 명령을 사용하여 테스트 사양 YAML 파일에 다음 명령을 추가합니다.

```
phases:
  install:
    commands:
      - export APPIUM_VERSION=2
      - npm install -g appium@$APPIUM_VERSION
```

`devicefarm-cli` 또는 다른 CLI 명령에 대한 자세한 내용은 AWS CLI 참조를 참조하세요.

프레임워크의 모든 기능(주석 등)을 사용하려면 사용자 지정 테스트 환경을 선택하고 AWS CLI 또는 `Device Farm` 콘솔을 사용하여 사용자 지정 테스트 사양을 업로드합니다.

주제

- Appium 테스트 패키지 구성
- 압축 테스트 패키지 파일 생성
- Device Farm에 테스트 패키지 업로드
- 테스트의 스크린샷 캡처(선택 사항)

## Appium 테스트 패키지 구성

다음 지침을 사용하여 테스트 패키지를 구성하세요.

Java (JUnit)

1. `pom.xml`을 수정하여 패키징을 JAR 파일로 설정하세요.

   ```
   <groupId>com.acme</groupId>
   <artifactId>acme-myApp-appium</artifactId>
   <version>1.0-SNAPSHOT</version>
   <packaging>jar</packaging>
   ```

2. `maven-jar-plugin`을 사용하도록 `pom.xml`을 수정하여 테스트를 JAR 파일에 빌드하세요.

다음 플러그인은 테스트 소스 코드(src/test 디렉터리에 포함된 모든 항목)를 JAR 파일에 빌드하세요.

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-jar-plugin</artifactId>
  <version>2.6</version>
  <executions>
    <execution>
      <goals>
        <goal>test-jar</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

3. maven-dependency-plugin을 사용하도록 pom.xml을 수정하여 종속성을 JAR 파일로 빌드하세요.

   다음 플러그인은 종속 항목을 dependency-jars 디렉터리에 복사합니다.

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-dependency-plugin</artifactId>
  <version>2.10</version>
  <executions>
    <execution>
      <id>copy-dependencies</id>
      <phase>package</phase>
      <goals>
        <goal>copy-dependencies</goal>
      </goals>
      <configuration>
        <outputDirectory>${project.build.directory}/dependency-jars/</
outputDirectory>
      </configuration>
    </execution>
  </executions>
</plugin>
```

4. 다음 XML 어셈블리를 src/main/assembly/zip.xml에 저장하세요.

다음 XML은 Maven에게 빌드 출력 디렉터리와 dependency-jars 디렉터리의 루트에 있는 모든 항목을 포함하는 .zip 파일을 빌드하도록 지시하는 어셈블리 정의입니다(구성된 경우에 해당).

```
<assembly
    xmlns="http://maven.apache.org/plugins/maven-assembly-plugin/assembly/1.1.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/plugins/maven-assembly-plugin/
assembly/1.1.0 http://maven.apache.org/xsd/assembly-1.1.0.xsd">
  <id>zip</id>
  <formats>
    <format>zip</format>
  </formats>
  <includeBaseDirectory>false</includeBaseDirectory>
  <fileSets>
    <fileSet>
      <directory>${project.build.directory}</directory>
      <outputDirectory>./</outputDirectory>
      <includes>
        <include>*.jar</include>
      </includes>
    </fileSet>
    <fileSet>
      <directory>${project.build.directory}</directory>
      <outputDirectory>./</outputDirectory>
      <includes>
        <include>/dependency-jars/</include>
      </includes>
    </fileSet>
  </fileSets>
</assembly>
```

5. maven-assembly-plugin을 사용하도록 pom.xml을 수정하여 테스트와 모든 종속성을 하나의 .zip 파일에 패키징합니다.

   다음 플러그인은 앞의 어셈블리를 사용하여 mvn package가 실행될 때마다 빌드 출력 디렉터리에 zip-with-dependencies이라는 .zip 파일을 만듭니다.

```
<plugin>
  <artifactId>maven-assembly-plugin</artifactId>
  <version>2.5.4</version>
  <executions>
```

```
    <execution>
      <phase>package</phase>
      <goals>
        <goal>single</goal>
      </goals>
      <configuration>
        <finalName>zip-with-dependencies</finalName>
        <appendAssemblyId>false</appendAssemblyId>
        <descriptors>
          <descriptor>src/main/assembly/zip.xml</descriptor>
        </descriptors>
      </configuration>
    </execution>
  </executions>
</plugin>
```

> **Note**
>
> 1.3에서 주석이 지원되지 않는다는 오류 메시지가 나타나면 pom.xml에 다음을 추가하세요.
>
> ```
> <plugin>
>   <artifactId>maven-compiler-plugin</artifactId>
>   <configuration>
>     <source>1.7</source>
>     <target>1.7</target>
>   </configuration>
> </plugin>
> ```

Java (TestNG)

1. pom.xml을 수정하여 패키징을 JAR 파일로 설정하세요.

   ```
   <groupId>com.acme</groupId>
   <artifactId>acme-myApp-appium</artifactId>
   <version>1.0-SNAPSHOT</version>
   <packaging>jar</packaging>
   ```

2. maven-jar-plugin을 사용하도록 pom.xml을 수정하여 테스트를 JAR 파일에 빌드하세요.

다음 플러그인은 테스트 소스 코드(src/test 디렉터리에 포함된 모든 항목)를 JAR 파일에 빌드하세요.

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-jar-plugin</artifactId>
  <version>2.6</version>
  <executions>
    <execution>
      <goals>
        <goal>test-jar</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

3. maven-dependency-plugin을 사용하도록 pom.xml을 수정하여 종속성을 JAR 파일로 빌드하세요.

   다음 플러그인은 종속 항목을 dependency-jars 디렉터리에 복사합니다.

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-dependency-plugin</artifactId>
  <version>2.10</version>
  <executions>
    <execution>
      <id>copy-dependencies</id>
      <phase>package</phase>
      <goals>
        <goal>copy-dependencies</goal>
      </goals>
      <configuration>
        <outputDirectory>${project.build.directory}/dependency-jars/</
outputDirectory>
      </configuration>
    </execution>
  </executions>
</plugin>
```

4. 다음 XML 어셈블리를 src/main/assembly/zip.xml에 저장하세요.

다음 XML은 Maven에게 빌드 출력 디렉터리와 dependency-jars 디렉터리의 루트에 있는 모든 항목을 포함하는 .zip 파일을 빌드하도록 지시하는 어셈블리 정의입니다(구성된 경우에 해당).

```
<assembly
    xmlns="http://maven.apache.org/plugins/maven-assembly-plugin/assembly/1.1.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/plugins/maven-assembly-plugin/
assembly/1.1.0 http://maven.apache.org/xsd/assembly-1.1.0.xsd">
  <id>zip</id>
  <formats>
    <format>zip</format>
  </formats>
  <includeBaseDirectory>false</includeBaseDirectory>
  <fileSets>
    <fileSet>
      <directory>${project.build.directory}</directory>
      <outputDirectory>./</outputDirectory>
      <includes>
        <include>*.jar</include>
      </includes>
    </fileSet>
    <fileSet>
      <directory>${project.build.directory}</directory>
      <outputDirectory>./</outputDirectory>
      <includes>
        <include>/dependency-jars/</include>
      </includes>
    </fileSet>
  </fileSets>
</assembly>
```

5. maven-assembly-plugin을 사용하도록 pom.xml을 수정하여 테스트와 모든 종속성을 하나의 .zip 파일에 패키징합니다.

   다음 플러그인은 앞의 어셈블리를 사용하여 mvn package가 실행될 때마다 빌드 출력 디렉터리에 zip-with-dependencies이라는 .zip 파일을 만듭니다.

```
<plugin>
  <artifactId>maven-assembly-plugin</artifactId>
  <version>2.5.4</version>
  <executions>
```

```
    <execution>
      <phase>package</phase>
      <goals>
        <goal>single</goal>
      </goals>
      <configuration>
        <finalName>zip-with-dependencies</finalName>
        <appendAssemblyId>false</appendAssemblyId>
        <descriptors>
          <descriptor>src/main/assembly/zip.xml</descriptor>
        </descriptors>
      </configuration>
    </execution>
  </executions>
</plugin>
```

> **Note**
>
> 1.3에서 주석이 지원되지 않는다는 오류 메시지가 나타나면 pom.xml에 다음을 추가하세요.
>
> ```
> <plugin>
>   <artifactId>maven-compiler-plugin</artifactId>
>   <configuration>
>     <source>1.7</source>
>     <target>1.7</target>
>   </configuration>
> </plugin>
> ```

### Node.JS

Appium Node.js 테스트를 패키징하여 Device Farm에 업로드하려면 로컬 컴퓨터에 다음을 설치해야 합니다.

- 노드 버전 매니저(nvm)

  불필요한 종속 항목이 테스트 패키지에 포함되지 않도록 테스트를 개발하고 패키징하려면 이 도구를 사용하세요.
- Node.js

- npm-bundle(전역 설치됨)

1. nvm이 있는지 확인합니다.

   ```
   command -v nvm
   ```

   출력에 nvm이 나타납니다.

   자세한 내용은 nvm GitHub on을 참조하십시오.
2. 이 명령을 실행하여 Node.js를 설치하세요.

   ```
   nvm install node
   ```

   다음과 같이 Node.js의 특정 버전을 지정할 수 있습니다.

   ```
   nvm install 11.4.0
   ```
3. 올바른 버전의 노드가 사용 중인지 확인합니다.

   ```
   node -v
   ```
4. npm-bundle 전 세계 설치

   ```
   npm install -g npm-bundle
   ```

Python

1. 불필요한 종속 항목이 앱 패키지에 포함되지 않도록 테스트를 개발하고 패키징하기 위해서는 Python virtualenv를 설정하는 것이 좋습니다.

   ```
   $ virtualenv workspace
   $ cd workspace
   $ source bin/activate
   ```

> **Tip**
>
> - `--system-site-packages` 옵션으로 Python virtualenv를 생성하지 마세요. 글로벌 사이트 패키지 디렉터리에서 패키지를 상속하기 때문입니다. 이로 인해 테스트에 필요하지 않은 종속성이 가상 환경에 포함될 수 있습니다.
> - 또한 이러한 테스트가 실행되는 인스턴스에 이러한 기본 라이브러리가 없을 수 있으므로, 테스트가 기본 라이브러리에 종속된 종속성을 사용하지 않음을 확인해야 합니다.

2. 가상 환경에 py.test를 설치하세요.

```
$ pip install pytest
```

3. 가상 환경에 Appium Python 클라이언트를 설치하세요.

```
$ pip install Appium-Python-Client
```

4. 사용자 지정 모드에서 다른 경로를 지정하지 않으면 Device Farmd은 테스트가 `tests/`에 저장될 것으로 예상합니다. `find`를 사용하여 폴더 내의 모든 파일을 표시할 수 있습니다.

```
$ find tests/
```

이러한 파일에 Device Farm에서 실행할 테스트 스위트가 포함되어 있는지 확인합니다.

```
tests/
tests/my-first-tests.py
tests/my-second-tests/py
```

5. 가상 환경 작업 공간 폴더에서 이 명령을 실행하여 실행 없이도 테스트 목록을 표시합니다.

```
$ py.test --collect-only tests/
```

출력에 Device Farm에서 실행하려는 테스트가 표시되는지 확인하세요.

6. 테스트/폴더 아래의 모든 캐시된 파일을 정리하세요.

```
$ find . -name '__pycache__' -type d -exec rm -r {} +
$ find . -name '*.pyc' -exec rm -f {} +
```

```
$ find . -name '*.pyo' -exec rm -f {} +
$ find . -name '*~' -exec rm -f {} +
```

7. 작업 공간에서 다음 명령을 실행하여 requirements.txt 파일을 생성하세요.

```
$ pip freeze > requirements.txt
```

Ruby

Appium Ruby 테스트를 패키징하여 Device Farm에 업로드하려면 로컬 컴퓨터에 다음을 설치해야 합니다.

- Ruby 버전 관리자(RVM)

  불필요한 종속 항목이 테스트 패키지에 포함되지 않도록 테스트를 개발하고 패키징하려면 이 명령줄 도구를 사용하세요.
- Ruby
- Bundler(이 gem은 일반적으로 Ruby와 함께 설치됩니다.)

1. 필수 키와 RVM 및 Ruby를 설치하세요. 자세한 내용은 RVM 웹 사이트에서 RVM 설치를 참조하세요.

   설치가 완료되면 로그아웃한 후 다시 로그인하여 터미널을 다시 로드합니다.

   > **Note**
   >
   > RVM이 bash 쉘 전용 함수로 로드됩니다.

2. rvm이 올바르게 설치되었는지 확인하세요.

```
command -v rvm
```

   출력에 `rvm`이 나타납니다.

3. 특정 버전의 Ruby(예: *2.5.3*)를 설치하려면 다음 명령을 실행하세요.

```
rvm install ruby 2.5.3 --autolibs=0
```

   요청된 버전의 Ruby를 사용하고 있는지 확인하세요.

```
ruby -v
```

4. 원하는 테스트 플랫폼용 패키지를 컴파일하도록 번들러를 구성하세요.

```
bundle config specific_platform true
```

5. .lock 파일을 업데이트하여 테스트를 실행하는 데 필요한 플랫폼을 추가하세요.

   - Android 디바이스에서 실행할 테스트를 컴파일하는 경우 다음 명령어를 실행하여 Android 테스트 호스트의 종속성을 사용하도록 Gemfile을 구성하세요.

     ```
     bundle lock --add-platform x86_64-linux
     ```

   - iOS 디바이스에서 실행할 테스트를 컴파일하는 경우 다음 명령을 실행하여 iOS 테스트 호스트에 대한 종속성을 사용하도록 Gemfile을 구성하세요.

     ```
     bundle lock --add-platform x86_64-darwin
     ```

6. bundler gem은 대개 기본적으로 설치됩니다. 설치되어 있지 않다면 다음을 설치하세요.

```
gem install bundler -v 2.3.26
```

## 압축 테스트 패키지 파일 생성

> **Warning**
>
> Device Farm에서는 압축된 테스트 패키지에 있는 파일의 폴더 구조가 중요하며 일부 아카이브 도구는 ZIP 파일의 구조를 암시적으로 변경합니다. 로컬 데스크톱의 파일 관리자에 내장된 아카이브 유틸리티(예: 파인더 또는 Windows 탐색기)를 사용하는 대신 아래에 지정된 명령줄 유틸리티를 따르는 것이 좋습니다.

이제 Device Farm에 대한 테스트를 번들하세요.

Java (JUnit)

테스트 빌드 및 패키징

```
$ mvn clean package -DskipTests=true
```

`zip-with-dependencies.zip` 파일이 결과로 생성됩니다. 이것은 테스트 패키지입니다.

Java (TestNG)

테스트 빌드 및 패키징

```
$ mvn clean package -DskipTests=true
```

`zip-with-dependencies.zip` 파일이 결과로 생성됩니다. 이것은 테스트 패키지입니다.

Node.JS

1. 프로젝트를 확인하세요.

   프로젝트의 root 디렉터리에 있어야 합니다. root 디렉터리에서 `package.json`을 볼 수 있습니다.

2. 이 명령을 실행하여 로컬 종속 항목을 설치합니다.

   ```
   npm install
   ```

   또한 이 명령은 현재 디렉터리에 `node_modules` 폴더를 만듭니다.

   > **Note**
   >
   > 이제 테스트를 로컬로 실행할 수 있습니다.

3. 이 명령을 실행하여 현재 폴더의 파일을 *.tgz 파일로 패키징하세요. 이 파일은 `package.json` 파일의 `name` 속성을 사용하여 이름이 지정됩니다.

   ```
   npm-bundle
   ```

   이 tarball(.tgz) 파일에는 모든 코드와 종속 항목이 들어 있습니다.

4. 이 명령을 사용하여, 전 단계에서 생성한 tarball(*.tgz 파일)을 하나의 압축 아카이브로 묶어 생성하세요.

   ```
   zip -r MyTests.zip *.tgz
   ```

이것이 다음 절차에 Device Farm에 업로드하는 `MyTests.zip` 파일입니다.

### Python

#### Python 2

pip를 사용하여 필요한 Python 패키지("wheelhouse"라고 불림)의 아카이브를 생성하세요.

```
$ pip wheel --wheel-dir wheelhouse -r requirements.txt
```

wheelhouse, 테스트 및 pip 요구 사항을 Device Farm용 zip 아카이브로 패키징하세요.

```
$ zip -r test_bundle.zip tests/ wheelhouse/ requirements.txt
```

#### Python 3

테스트 및 pip 요구 사항을 zip 파일로 패키징하세요.

```
$ zip -r test_bundle.zip tests/ requirements.txt
```

### Ruby

1. 다음 명령을 실행하여 가상 Ruby 환경을 생성하세요.

   ```
   # myGemset is the name of your virtual Ruby environment
   rvm gemset create myGemset
   ```

2. 다음 명령을 실행하여 앞에서 생성한 환경을 사용하세요.

   ```
   rvm gemset use myGemset
   ```

3. 소스 코드를 확인하세요.

   프로젝트의 root 디렉터리에 있어야 합니다. root 디렉터리에서 `Gemfile`을 볼 수 있습니다.

4. 이 명령을 실행하여 `Gemfile`의 로컬 종속 항목과 모든 gem을 설치하세요.

   ```
   bundle install
   ```

> **Note**
>
> 이제 테스트를 로컬로 실행할 수 있습니다. 다음 명령을 실행하여 테스트를 로컬로 실행하세요.
>
> ```
> bundle exec $test_command
> ```

5. `vendor/cache` 폴더의 gem을 패키징하세요.

```
# This will copy all the .gem files needed to run your tests into the vendor/
cache directory
bundle package --all-platforms
```

6. 다음 명령을 실행하여, 소스 코드와 모든 종속 항목을 하나의 압축 아카이브로 묶어 생성하세요.

```
zip -r MyTests.zip Gemfile vendor/ $(any other source code directory files)
```

이것이 다음 절차에 Device Farm에 업로드하는 `MyTests.zip` 파일입니다.

## Device Farm에 테스트 패키지 업로드

Device Farm 콘솔을 사용하여 테스트를 업로드할 수 있습니다.

1. https://console.aws.amazon.com/devicefarm에서 Device Farm 콘솔에 로그인하세요.
2. Device Farm 탐색 패널에서 모바일 디바이스 테스트를 선택한 다음 프로젝트를 선택하세요.
3. 새 사용자인 경우 새 프로젝트를 선택하고 프로젝트 이름을 입력한 다음 제출을 선택하세요.

   프로젝트가 이미 있는 경우 해당 프로젝트를 선택하여 테스트를 업로드할 수 있습니다.
4. 프로젝트를 연 후 새 실행 생성을 선택하세요.
5. 기본 Android 및 iOS 테스트의 경우

   애플리케이션 선택 페이지에서 모바일 앱을 선택하고 파일 선택을 클릭하여 애플리케이션의 배포 가능 패키지를 업로드하세요.

> **Note**
>
> 파일은 Android `.apk` 또는 iOS `.ipa`여야 합니다. iOS 애플리케이션은 시뮬레이터가 아닌 실제 디바이스용으로 빌드되어야합니다.

모바일 웹 애플리케이션 테스트의 경우

애플리케이션 선택 페이지에서 웹 앱을 선택하세요.

6. 테스트에 적절한 이름을 지정하세요. 여기에는 공백이나 구두점 조합이 포함될 수 있습니다.
7. 다음을 선택하세요.
8. 구성 페이지의 테스트 프레임워크 설정 섹션에서 Appium **##**를 선택한 다음 파일 선택을 하세요.
9. 테스트가 포함된 .zip 파일을 찾아서 선택하세요. 이 .zip 파일은 Appium 테스트 패키지 구성에 설명된 형식이어야 합니다.
10. 사용자 지정 환경에서 테스트 실행을 선택하세요. 이러한 실행 환경을 사용하면 테스트 설정, 해제 및 간접 호출을 완벽하게 제어할 수 있을 뿐만 아니라 특정 버전의 런타임 및 Appium 서버를 선택할 수 있습니다. 테스트 사양 파일을 통해 사용자 지정 환경을 구성할 수 있습니다. 자세한 내용은 AWS Device Farm에서 사용자 지정 테스트 환경으로 작업 섹션을 참조하십시오.
11. 다음을 선택한 다음 지침에 따라 디바이스를 선택하고 실행을 시작하세요. 자세한 내용은 Device Farm에서 테스트 실행 생성 단원을 참조하세요.

> **Note**
>
> Device Farm은 Appium 테스트를 수정하지 않습니다.

## 테스트의 스크린샷 캡처(선택 사항)

테스트의 일부로 스크린샷을 캡처할 수 있습니다.

Device Farm은 Appium 스크린샷 저장 위치로 예상하는 로컬 파일 시스템의 정규화된 경로로 `DEVICEFARM_SCREENSHOT_PATH` 속성을 설정합니다. 스크린샷이 저장되는 테스트별 디렉터리는 런타임에 정의됩니다. 스크린샷은 Device Farm 보고서에 자동으로 포함됩니다. 스크린샷을 보려면 Device Farm 콘솔에서 스크린샷 섹션을 선택하세요.

Appium 테스트에서 스크린샷을 캡처하는 자세한 방법은 Appium API 설명서의 스크린샷 캡처을 참조하세요.

# AWS Device Farm에서 Android 테스트 사용

Device Farm은 Android 디바이스에 대한 여러 가지 자동화 테스트 유형과 두 가지 기본 제공 테스트를 지원합니다.

## Android 애플리케이션 테스트 프레임워크

Android 디바이스에서 사용할 수 있는 테스트는 다음과 같습니다.

- Appium 및 AWS Device Farm과 함께 작업
- Android 및 AWS Device Farm용 계측 작업

## Android용 기본 제공 테스트 유형

Android 디바이스에 사용할 수 있는 기본 제공 테스트 유형에는 한 가지가 있습니다.

- 내장: fuzz (Android 및 iOS)

## Android 및 AWS Device Farm용 계측 작업

Device Farm은 Android의 계측 기능(JUnit, Espresso, Robotium 또는 계측 기반 테스트)에 대한 지원을 제공합니다.

Device Farm은 또한 샘플 Android 애플리케이션 및 계측(Espresso)을 포함한 세 가지 Android 자동화 프레임워크의 작동 테스트에 대한 링크를 제공합니다. 안드로이드용 Device Farm 샘플 앱은 에서 다운로드할 수 GitHub 있습니다.

주제

- 계측이란 무엇인가요?
- Android 계측 테스트 업로드
- Android 계측 테스트의 스크린샷 생성
- Android 계측 테스트를 위한 추가 고려 사항
- 스탠다드 모드 테스트 파싱

## 계측이란 무엇인가요?

Android 계측을 사용하면 테스트 코드에서 콜백 메서드를 호출할 수 있습니다. 따라서 마치 구성 요소를 디버깅하는 것처럼 구성 요소의 수명 주기를 단계별로 실행할 수 있습니다. 자세한 내용은 Android 개발자 도구 문서의 테스트 유형 및 위치 섹션에서 계측 테스트를 참조하세요.

## Android 계측 테스트 업로드

Device Farm 콘솔을 사용하여 테스트를 업로드하세요.

1. https://console.aws.amazon.com/devicefarm에서 Device Farm 콘솔에 로그인하세요.
2. Device Farm 탐색 패널에서 모바일 디바이스 테스트를 선택한 다음 프로젝트를 선택하세요.
3. 프로젝트 목록에서 테스트를 업로드할 프로젝트를 선택하세요.

> **Tip**
>
> 검색 창을 사용하여 인스턴스 목록을 이름별로 필터링할 수 있습니다.
> 프로젝트를 생성하려면, AWS Device Farm에서 프로젝트 생성의 지침을 따르세요.

4. 새 실행 생성 버튼이 표시되면, 클릭하세요.
5. 애플리케이션 선택 페이지에서 파일 선택을 선택하세요.
6. Android 앱 파일을 찾아 선택하세요. 파일은 .apk 파일이어야 합니다.
7. 다음을 선택하세요.
8. 구성 페이지의 테스트 프레임워크 설정 섹션에서 계측을 선택한 후 파일 선택을 선택하세요.
9. 테스트가 포함된 .apk 파일을 찾아서 선택하세요.
10. 다음을 선택한 후 디바이스를 선택하기 위한 나머지 지침을 완료하여 디바이스를 선택하고 실행을 시작하세요.

## Android 계측 테스트의 스크린샷 생성

Android 계측 테스트의 일부로 스크린샷을 캡처할 수 있습니다.

스크린샷을 캡처하려면 다음 메서드 중 하나를 호출하세요.

- Robotium의 경우 `takeScreenShot` 메서드(예: `solo.takeScreenShot();`)를 호출하세요.
- Spoon의 경우 `screenshot` 메서드를 호출하며 다음 예시를 참고하세요.

```
Spoon.screenshot(activity, "initial_state");
/* Normal test code... */
Spoon.screenshot(activity, "after_login");
```

테스트가 실행되는 동안 Device Farm은 스크린샷 파일이 있는 경우, 디바이스의 다음 위치에서 스크린샷을 가져온 후 테스트 보고서에 추가합니다.

- `/sdcard/robotium-screenshots`
- `/sdcard/test-screenshots`
- `/sdcard/Download/spoon-screenshots/test-class-name/test-method-name`
- `/data/data/application-package-name/app_spoon-screenshots/test-class-name/test-method-name`

## Android 계측 테스트를 위한 추가 고려 사항

시스템 애니메이션

Android Espresso 테스트 설명서에 따르면 실제 디바이스에서 테스트할 때는 시스템 애니메이션을 끄는 것이 좋습니다. Device Farm은 UnitRunnerAndroid.support.test.runner.AndroidJ 계측 테스트 러너를 사용하여 실행할 때 창 애니메이션 스케일, 트랜지션 애니메이션 스케일 및 애니메이터 지속 시간 스케일 설정을 자동으로 비활성화합니다.

테스트 기록자

Device Farm은 Robotium과 같이 record-and-playback 스크립팅 도구가 있는 프레임워크를 지원합니다.

## 스탠다드 모드 테스트 파싱

표준 실행 모드에서 Device Farm은 테스트 스위트를 구문 분석하고 실행할 고유한 테스트 클래스와 메서드를 식별합니다. 이는 Dex Test Parser라는 도구를 통해 수행됩니다.

Android 계측 .apk 파일을 입력으로 제공하면 파서는 JUnit 3 및 JUnit 4 규칙과 일치하는 테스트의 정규화된 메서드 이름을 반환합니다.

로컬 환경에서 이를 테스트하려면 다음을 참조하세요.

1. `dex-test-parser` 바이너리를 다운로드하세요.

2. 다음 명령을 실행하여 Device Farm에서 실행할 테스트 메서드의 목록을 가져오세요.

```
java -jar parser.jar path/to/apk path/for/output
```

# AWS Device Farm에서 iOS 테스트 사용

Device Farm은 iOS 디바이스에 대한 여러 가지 자동화 테스트 유형과 기본 제공 테스트를 지원합니다.

## iOS 애플리케이션 테스트 프레임워크

iOS 디바이스에 사용할 수 있는 테스트는 다음과 같습니다.

- Appium 및 AWS Device Farm과 함께 작업
- iOS용 XCTest 및 AWS Device Farm과 함께 작업
- XCTest UI

## iOS용 기본 제공 테스트 유형

현재 iOS 디바이스에 사용할 수 있는 테스트 유형은 한 가지 내장되어 있습니다.

- 내장: fuzz (Android 및 iOS)

## iOS용 XCTest 및 AWS Device Farm과 함께 작업

Device Farm을 사용하면 XCTest 프레임워크를 사용하여 실제 디바이스에서 앱을 테스트할 수 있습니다. XCTest에 대한 자세한 내용은 Xcode를 사용한 테스트의 테스트 기본 사항을 참조하세요.

테스트를 실행하려면 테스트 실행용 패키지를 만들고 이 패키지를 Device Farm에 업로드하세요.

주제

- XCTest 실행을 위한 패키지 생성
- XCTest 실행을 위한 패키지를 Device Farm에 업로드

### XCTest 실행을 위한 패키지 생성

XCTest 프레임워크를 사용하여 앱을 테스트하려면 Device Farm에 다음이 필요합니다.

- `.ipa` 파일인 앱 패키지
- `.zip` 파일인 XCTest 패키지

Xcode가 생성하는 빌드 출력을 사용하여 이러한 패키지를 생성하세요. Device Farm에 업로드할 수 있도록 패키지를 만들려면 다음 단계를 완료하세요.

앱의 빌드 출력을 생성하려면 다음을 참고하세요.

1. Xcode에서 앱 프로젝트를 여세요.
2. Xcode 툴바의 구성표 드롭다운 메뉴에서 일반 iOS 디바이스를 대상으로 선택하세요.
3. 제품 메뉴에서 다음에 대한 구축를 선택한 다음 테스트를 선택하세요.

### 앱 패키지 생성

1. Xcode의 프로젝트 탐색기의 제품에서 이름이 *app-project-name*.app로 지정된 파일의 컨텍스트 메뉴를 여세요. 그런 다음 파인더에서 보기를 선택하세요. 파인더에서 Xcode가 테스트 빌드를 위해 생성한 출력이 포함된 Debug-iphoneos라는 이름의 폴더가 열립니다. 이 폴더에는 .app 파일이 포함되어 있습니다.
2. 파인더에서 새 폴더를 만들고 이름을 Payload로 지정합니다.
3. *app-project-name*.app 파일을 복사하여 Payload 폴더에 붙여 넣습니다.
4. Payload 폴더의 컨텍스트 메뉴를 열고 “Payload” 압축을 선택하세요. Payload.zip이라는 이름의 파일이 생성됩니다.
5. Payload.zip의 파일 이름과 확장자를 *app-project-name*.ipa로 변경하세요.

   이후 단계에서 이 파일을 Device Farm에 제공합니다. 파일을 더 쉽게 찾을 수 있도록 데스크톱과 같은 다른 위치로 옮기는 것이 좋습니다.
6. 원하는 경우 Payload 폴더와 그 안에 있는 .app 파일을 삭제할 수 있습니다.

### XCTest 패키지 생성

1. 파인더에서 Debug-iphoneos 디렉터리 내 *app-project-name*.app 파일의 컨텍스트 메뉴를 여세요. 그런 다음 패키지 콘텐츠 보기를 선택하세요.
2. 패키지 콘텐츠에서 Plugins 폴더를 여세요. 이 폴더에 *app-project-name*.xctest 파일이 포함되어 있습니다.

3. 이 파일의 컨텍스트 메뉴를 열고 “***app-project-name*.xctest**” 압축을 선택하세요. *app-project-name*.xctest.zip이라는 파일이 생성됩니다.

   이후 단계에서 이 파일을 Device Farm에 제공합니다. 파일을 더 쉽게 찾을 수 있도록 데스크톱과 같은 다른 위치로 옮기는 것이 좋습니다.

## XCTest 실행을 위한 패키지를 Device Farm에 업로드

Device Farm 콘솔을 사용하여 테스트 패키지를 업로드합니다.

1. https://console.aws.amazon.com/devicefarm에서 Device Farm 콘솔에 로그인하세요.
2. 프로젝트가 없는 경우 생성하세요. 프로젝트를 만드는 단계는 AWS Device Farm에서 프로젝트 생성 단원을 참조하세요.

   그렇지 않으면 Device Farm 탐색 패널에서 모바일 디바이스 테스트를 선택한 다음 프로젝트를 선택하세요.
3. 테스트를 실행하는 데 사용할 프로젝트를 선택하세요.
4. 새 실행 생성을 선택하세요.
5. 애플리케이션 선택 페이지에서 모바일 앱을 선택하세요.
6. 파일 선택을 선택하세요.
7. `.ipa` 파일을 찾고 업로드합니다.

   > **Note**
   >
   > 테스트를 위해 `.ipa` 패키지를 빌드해야 합니다.

8. 업로드가 완료되면 다음을 선택하세요.
9. 구성 페이지의 테스트 프레임워크 설정 섹션에서 XCTest를 선택하세요. 파일 선택을 선택하세요.
10. 앱의 XCTest 패키지가 포함된 `.zip` 파일을 찾아 업로드하세요.
11. 업로드가 완료되면 다음을 선택하세요.
12. 프로젝트 생성 프로세스의 나머지 단계를 완료하여 설치하세요. 테스트할 디바이스를 선택하고 디바이스 상태를 지정하세요.
13. 검토 및 실행 시작 페이지에서 확인 및 실행 시작을 선택하세요.

    Device Farm은 테스트를 실행하고 콘솔에 결과를 표시합니다.

# iOS용 XCTest UI 테스트 프레임워크 및 AWS Device Farm과 함께 작업

Device Farm은 iOS용 XCTest UI 테스트 프레임워크를 지원합니다. 특히, Device Farm은 Objective-C와 Swift로 작성된 XCTest UI 테스트를 지원합니다.

주제

- XCTest UI 테스트 프레임워크란 무엇인가요?
- iOS XCTest UI 테스트 준비
- iOS XCTest UI 테스트 업로드
- iOS XCTest UI 테스트에서 스크린샷 생성

## XCTest UI 테스트 프레임워크란 무엇인가요?

XCTest UI 프레임워크는 Xcode 7에 도입된 새로운 테스트 프레임워크입니다. 이 프레임워크는 XCTest를 UI 테스트 기능으로 확장합니다. 자세한 내용은 iOS 개발자 라이브러리의 사용자 인터페이스 테스트를 참조하세요.

## iOS XCTest UI 테스트 준비

iOS XCTest UI 테스트 러너 번들은 올바른 형식의 .ipa 파일에 포함되어야 합니다.

.ipa 파일을 만들려면 my-project-name UITEST-Runner.app 번들을 빈 페이로드 디렉터리에 넣으십시오. 그런 다음 Payload 디렉터리를 .zip 파일에 보관한 다음 파일 확장자를 .ipa로 변경하세요. *UITest-Runner.app 번들은 테스트용 프로젝트를 빌드할 때 Xcode에서 생성합니다. 프로젝트의 제품 디렉터리에서 찾을 수 있습니다.

## iOS XCTest UI 테스트 업로드

Device Farm 콘솔을 사용하여 테스트를 업로드하세요.

1. https://console.aws.amazon.com/devicefarm에서 Device Farm 콘솔에 로그인하세요.
2. Device Farm 탐색 패널에서 모바일 디바이스 테스트를 선택한 다음 프로젝트를 선택하세요.
3. 프로젝트 목록에서 테스트를 업로드할 프로젝트를 선택하세요.

> **Tip**
>
> 검색 창을 사용하여 인스턴스 목록을 이름별로 필터링할 수 있습니다.

프로젝트를 생성하려면, AWS Device Farm에서 프로젝트 생성의 지침을 따르세요.

4. 새 실행 생성 버튼이 표시되면, 클릭하세요.
5. 애플리케이션 선택 페이지에서 파일 선택을 선택하세요.
6. Android 또는 iOS 앱 파일을 찾아 선택하세요. 파일은 .ipa 파일이어야 합니다.

> **Note**
>
> .ipa 파일은 시뮬레이터가 아닌 iOS 디바이스용으로 빌드되어야 합니다.

7. 다음을 선택하세요.
8. 구성 페이지의 테스트 프레임워크 설정 섹션에서 XCTest UI를 선택한 다음 파일 선택을 선택하세요.
9. iOS XCTest UI 테스트 러너가 포함된 .ipa 파일을 찾아 선택하세요.
10. 다음을 선택한 후 나머지 지침을 완료하여 디바이스를 선택하고 실행을 시작합니다.

### iOS XCTest UI 테스트에서 스크린샷 생성

XCTest UI 테스트는 테스트의 모든 단계에서 스크린샷을 자동으로 캡처합니다. 이 스크린샷은 Device Farm 테스트 보고서에 표시됩니다. 추가 코드가 필요하지 않습니다.

## AWS Device Farm에서 웹 앱 테스트 사용

Device Farm은 Appium을 사용하여 웹 애플리케이션을 테스트하는 기능을 제공합니다. Device Farm에서 Appium 테스트를 설정하는 방법에 대한 자세한 내용은 the section called “Appium”을 참고하세요.

### 측정된 디바이스 및 측정되지 않은 디바이스에 대한 규칙

측정이란 디바이스에 대한 요금 청구를 말합니다. 기본적으로 Device Farm 디바이스는 미터링 측정되며 무료 평가판 사용 시간이 모두 소진되면 분당 요금이 부과됩니다. 또한 무제한 디바이스를 구매하여 월 정액 요금으로 무제한 테스트를 수행할 수 있습니다. 요금에 대한 자세한 내용은 AWS Device Farm 요금을 참조하세요.

iOS 디바이스와 Android 디바이스가 모두 포함된 디바이스 풀에서 실행을 시작하기로 선택한 경우 측정된 디바이스와 측정되지 않은 디바이스에 대한 규칙이 있습니다. 예를 들어 무제한 Android 디바이스 5대와 무제한 iOS 디바이스 5대가 있는 경우 웹 테스트 실행은 무제한 디바이스를 사용합니다.

또 다른 예는 다음과 같습니다. 무제한 Android 디바이스가 5대 있고 무제한 iOS 디바이스는 없다고 가정합니다. 웹 실행에 Android 디바이스만 선택하면 무제한 디바이스가 사용됩니다. 웹 실행에 Android와 iOS 디바이스를 모두 선택하면 청구 방법에 따라 측정되며 무제한 디바이스는 사용되지 않습니다.

# AWS Device Farm에서 내장된 테스트로 작업

Device Farm은 Android 및 iOS 디바이스에 내장된 테스트 유형을 지원합니다.

## 기본 제공 테스트 유형

기본 제공 테스트를 사용하면 스크립트를 작성하지 않고도 앱을 테스트할 수 있습니다.

- 내장: fuzz (Android 및 iOS)

## Device Farm에 내장된 fuzz 테스트 사용하기

Device Farm은 내장된 fuzz 테스트 유형을 제공합니다.

### 기본 제공 fuzz 테스트란 무엇인가요?

내장된 fuzz 테스트는 사용자 인터페이스 이벤트를 디바이스에 무작위로 전송한 다음 결과를 보고합니다.

### 기본 제공 fuzz 테스트 유형 사용

Device Farm 콘솔을 사용하여 내장된 fuzz 테스트를 실행합니다.

1. https://console.aws.amazon.com/devicefarm에서 Device Farm 콘솔에 로그인하세요.
2. Device Farm 탐색 패널에서 모바일 디바이스 테스트를 선택한 다음 프로젝트를 선택하세요.
3. 프로젝트 목록에서 내장된 fuzz 테스트를 실행할 프로젝트를 선택하세요.

   > **Tip**
   > 검색 창을 사용하여 인스턴스 목록을 이름별로 필터링할 수 있습니다.
   > 프로젝트를 생성하려면, AWS Device Farm에서 프로젝트 생성의 지침을 따르세요.

4. 새 실행 생성 버튼이 표시되면, 클릭하세요.
5. 애플리케이션 선택 페이지에서 파일 선택을 선택하세요.

6. 내장된 fuzz 테스트를 실행할 앱 파일을 찾아 선택하세요.
7. 다음을 선택하세요.
8. 구성 페이지의 설정 테스트 프레임워크 섹션에서 내장: Fuzz를 선택하세요.
9. 다음 설정 중 하나가 나타나면 기본값을 그대로 사용하거나 직접 지정할 수 있습니다.

   - 이벤트 개수: fuzz 테스트가 수행할 사용자 인터페이스 이벤트 수를 나타내는 1~10,000 사이의 숫자를 지정합니다.
   - 이벤트 스로틀: 다음 사용자 인터페이스 이벤트를 수행하기 전에 fuzz 테스트가 대기할 시간을 밀리초 단위로 나타내는 0~1,000 사이의 숫자를 지정합니다.
   - Randomizer 시드: 사용자 인터페이스 이벤트를 무작위화하는 데 사용할 fuzz 테스트의 숫자를 지정합니다. 후속 fuzz 테스트에 동일한 숫자를 지정하면 이벤트 시퀀스가 동일하도록 할 수 있습니다.

10. 다음을 선택한 후 디바이스를 선택하기 위한 나머지 지침을 완료하여 디바이스를 선택하고 실행을 시작하세요.

# AWS Device Farm에서 사용자 지정 테스트 환경으로 작업

AWS Device Farm을 사용하면 모든 Device Farm 사용자에게 권장되는 접근 방식인 자동 테스트(사용자 지정 모드)를 위한 사용자 지정 환경을 구성할 수 있습니다. Device Farm의 환경에 대해 자세히 알아보려면 테스트 환경을 참조하세요.

표준 모드와 달리 사용자 지정 모드의 이점은 다음과 같습니다.

- 더 빠른 end-to-end 테스트 실행: 테스트 패키지가 테스트 도구 모음의 모든 테스트를 탐지하도록 파싱되지 않으므로 사전 처리/사후 처리 오버헤드를 피할 수 있습니다.
- 라이브 로그 및 비디오 스트리밍: 사용자 지정 모드를 사용하면 클라이언트 측 테스트 로그와 비디오가 실시간 스트리밍됩니다. 이 기능은 표준 모드의 경우 사용할 수 없습니다.
- 모든 아티팩트 캡처: 호스트 및 장치에서 사용자 지정 모드를 사용하면 모든 테스트 아티팩트를 캡처할 수 있습니다. 표준 모드에서는 불가능할 수 있습니다.
- 보다 일관되고 복제 가능한 로컬 환경: 표준 모드에서는 개별 테스트마다 아티팩트가 별도로 제공되므로 특정 상황에서 유용할 수 있습니다. 하지만 Device Farm이 실행된 각 테스트를 다르게 처리하므로 로컬 테스트 환경이 원래 구성과 다를 수 있습니다.

  반면 사용자 지정 모드를 사용하면 Device Farm 테스트 실행 환경을 로컬 테스트 환경에 맞게 일관되게 만들 수 있습니다.

사용자 지정 환경은 YAML 형식의 테스트 사양(테스트 사양) 파일을 사용하여 구성됩니다. Device Farm은 지원되는 각 테스트 유형에 대해 있는 그대로 사용하거나 사용자 지정할 수 있는 기본 테스트 사양 파일을 제공합니다. 테스트 필터 또는 구성 파일과 같은 사용자 지정을 테스트 사양에 추가할 수 있습니다. 편집된 테스트 사양은 향후 테스트 실행을 위해 저장할 수 있습니다.

자세한 정보는 AWS CLI를 사용하여 사용자 지정 테스트 사양 업로드 및 Device Farm에서 테스트 실행 생성 단원을 참조하세요.

주제

# 테스트 사양 구문

YAML 테스트 사양 파일 구조

```
version: 0.1

phases:
  install:
    commands:
      - command
      - command
  pre_test:
    commands:
      - command
      - command
  test:
    commands:
      - command
      - command
  post_test:
    commands:
      - command
      - command

artifacts:
  - location
  - location
```

테스트 사양에는 다음이 포함되어 있습니다.

**`version`**

Device Farm에서 지원하는 테스트 사양 버전을 반영합니다. 현재 버전 번호는 0.1입니다.

**`phases`**

이 섹션에는 테스트 실행 중에 실행되는 명령 그룹이 포함되어 있습니다.

허용되는 테스트 단계 이름은 다음과 같습니다.

**`install`**

선택 사항

Device Farm에서 지원되는 프레임워크를 테스트하기 위한 기본 종속 항목이 이미 설치되어 있습니다. 이 단계에는 Device Farm이 설치 중에 실행하는 추가 명령(있는 경우)이 포함되어 있습니다.

**`pre_test`**

선택 사항

자동 테스트 실행 전에 실행되는 명령이 있는 경우입니다.

**`test`**

선택 사항

자동 테스트 실행 중에 실행되는 명령입니다. 테스트 단계에서 명령이 실패할 경우 테스트가 실패로 표시됩니다.

**`post_test`**

선택 사항

자동 테스트 실행 후에 실행되는 명령이 있는 경우입니다.

**`artifacts`**

선택 사항

Device Farm은 여기에 지정된 위치에서 사용자 지정 보고서, 로그 파일 및 이미지 등의 아티팩트를 수집합니다. 와일드카드 문자는 아티팩트 위치의 일부로 지원되지 않으므로 각 위치의 올바른 경로를 지정해야 합니다.

이러한 테스트 아티팩트는 테스트 실행에서 각 디바이스에 사용할 수 있습니다. 테스트 아티팩트 검색에 대한 자세한 내용은 사용자 지정 테스트 환경에서 아티팩트 사용 단원을 참조하세요.

**Important**

테스트 사양은 올바른 YAML 파일 형식으로 지정해야 합니다. 테스트 사양에서 들여쓰기 또는 공백이 잘못된 경우 테스트 실행이 실패할 수 있습니다. YAML 파일에서 탭은 허용되지 않습니다. YAML 검사기를 사용하여 테스트 사양이 올바른 YAML 파일인지 여부를 테스트할 수 있습니다. 자세한 내용은 YAML 웹 사이트를 참조하세요.

# 테스트 사양 예

Appium Java TestNG 테스트 실행으로 구성하는 Device Farm YAML 테스트 사양의 예입니다.

```
version: 0.1

# This flag enables your test to run using Device Farm's Amazon Linux 2 test host when
 scheduled on
# Android devices. By default, iOS device tests will always run on Device Farm's macOS
 test hosts.
# For Android, you can explicitly select your test host to use our Amazon Linux 2
 infrastructure.
# For more information, please see:
# https://docs.aws.amazon.com/devicefarm/latest/developerguide/amazon-linux-2.html
android_test_host: amazon_linux_2

# Phases represent collections of commands that are executed during your test run on
 the test host.
phases:

  # The install phase contains commands for installing dependencies to run your tests.
  # For your convenience, certain dependencies are preinstalled on the test host.

  # For Android tests running on the Amazon Linux 2 test host, many software libraries
 are available
  # from the test host using the devicefarm-cli tool. To learn more, please see:
  # https://docs.aws.amazon.com/devicefarm/latest/developerguide/amazon-linux-2-
devicefarm-cli.html

  # For iOS tests, you can use the Node.JS tools nvm, npm, and avm to setup your
 environment. By
  # default, Node.js versions 16.20.2 and 14.19.3 are available on the test host.
  install:
    commands:
      # The Appium server is written using Node.js. In order to run your desired
 version of Appium,
      # you first need to set up a Node.js environment that is compatible with your
 version of Appium.
      - |-
        if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "Android" ];
        then
          devicefarm-cli use node 16;
        else
```

```
        # For iOS, use "nvm use" to switch between the two preinstalled NodeJS
versions 14 and 16,
        # and use "nvm install" to download a new version of your choice.
        nvm use 16;
      fi;
    - node --version

    # Use the devicefarm-cli to select a preinstalled major version of Appium on
Android.
    # Use avm or npm to select Appium for iOS.
    - |-
      if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "Android" ];
      then
        # For Android, the Device Farm service automatically updates the preinstalled
Appium versions
        # over time to incorporate the latest minor and patch versions for each major
version. If you
        # wish to select a specific version of Appium, you can instead use NPM to
install it:
        # npm install -g appium@2.1.3;
        devicefarm-cli use appium 2;
      else
        # For iOS, Appium versions 1.22.2 and 2.2.1 are preinstalled and selectable
through avm.
        # For all other versions, please use npm to install them. For example:
        # npm install -g appium@2.1.3;
        # Note that, for iOS devices, Appium 2 is only supported on iOS version 14
and above using
        # NodeJS version 16 and above.
        avm 2.2.1;
      fi;
    - appium --version

    # For Appium version 2, for Android tests, Device Farm automatically updates the
preinstalled
    # UIAutomator2 driver over time to incorporate the latest minor and patch
versions for its major
    # version 2. If you want to install a specific version of the driver, you can use
the Appium
    # extension CLI to uninstall the existing UIAutomator2 driver and install your
desired version:
    # - |-
    #   if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "Android" ];
    #   then
```

```
      #     appium driver uninstall uiautomator2;
      #     appium driver install uiautomator2@2.34.0;
      #   fi;

      # For Appium version 2, for iOS tests, the XCUITest driver is preinstalled using
version 5.7.0
      # If you want to install a different version of the driver, you can use the
Appium extension CLI
      # to uninstall the existing XCUITest driver and install your desired version:
      # - |-
      #   if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "iOS" ];
      #   then
      #     appium driver uninstall xcuitest;
      #     appium driver install xcuitest@5.8.1;
      #   fi;

      # We recommend setting the Appium server's base path explicitly for accepting
commands.
      - export APPIUM_BASE_PATH=/wd/hub

      # Install the NodeJS dependencies.
      - cd $DEVICEFARM_TEST_PACKAGE_PATH
      # First, install dependencies which were packaged with the test package using
npm-bundle.
      - npm install *.tgz
      # Then, optionally, install any additional dependencies using npm install.
      # If you do run these commands, we strongly recommend that you include your
package-lock.json
      # file with your test package so that the dependencies installed on Device Farm
match
      # the dependencies you've installed locally.
      # - cd node_modules/*
      # - npm install

  # The pre-test phase contains commands for setting up your test environment.
  pre_test:
    commands:
      # Device farm provides different pre-built versions of WebDriverAgent, an
essential Appium
      # dependency for iOS devices, and each version is suggested for different
versions of Appium:
      # DEVICEFARM_WDA_DERIVED_DATA_PATH_V8: this version is suggested for Appium 2
      # DEVICEFARM_WDA_DERIVED_DATA_PATH_V7: this version is suggested for Appium 1
```

```
      # Additionally, for iOS versions 16 and below, the device unique identifier
 (UDID) needs
      # to be slightly modified for Appium tests.
      - |-
        if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "iOS" ];
        then
          if [ $(appium --version | cut -d "." -f1) -ge 2 ];
          then
            DEVICEFARM_WDA_DERIVED_DATA_PATH=$DEVICEFARM_WDA_DERIVED_DATA_PATH_V8;
          else
            DEVICEFARM_WDA_DERIVED_DATA_PATH=$DEVICEFARM_WDA_DERIVED_DATA_PATH_V7;
          fi;

          if [ $(echo $DEVICEFARM_DEVICE_OS_VERSION | cut -d "." -f 1) -le 16 ];
          then
            DEVICEFARM_DEVICE_UDID_FOR_APPIUM=$(echo $DEVICEFARM_DEVICE_UDID | tr -d
 "-");
          else
            DEVICEFARM_DEVICE_UDID_FOR_APPIUM=$DEVICEFARM_DEVICE_UDID;
          fi;
        fi;

      # Appium downloads Chromedriver using a feature that is considered insecure for
 multitenant
      # environments. This is not a problem for Device Farm because each test host is
 allocated
      # exclusively for one customer, then terminated entirely. For more information,
 please see
      # https://github.com/appium/appium/blob/master/packages/appium/docs/en/guides/
security.md

      # We recommend starting the Appium server process in the background using the
 command below.
      # The Appium server log will be written to the $DEVICEFARM_LOG_DIR directory.
      # The environment variables passed as capabilities to the server will be
 automatically assigned
      # during your test run based on your test's specific device.
      # For more information about which environment variables are set and how they're
 set, please see
      # https://docs.aws.amazon.com/devicefarm/latest/developerguide/custom-test-
environment-variables.html
      - |-
        if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "Android" ];
        then
```

```
          appium --base-path=$APPIUM_BASE_PATH --log-timestamp \
            --log-no-colors --relaxed-security --default-capabilities \
            "{\"appium:deviceName\": \"$DEVICEFARM_DEVICE_NAME\", \
            \"platformName\": \"$DEVICEFARM_DEVICE_PLATFORM_NAME\", \
            \"appium:app\": \"$DEVICEFARM_APP_PATH\", \
            \"appium:udid\":\"$DEVICEFARM_DEVICE_UDID\", \
            \"appium:platformVersion\": \"$DEVICEFARM_DEVICE_OS_VERSION\", \
            \"appium:chromedriverExecutableDir\":
\"$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR\", \
            \"appium:automationName\": \"UiAutomator2\"}" \
            >> $DEVICEFARM_LOG_DIR/appium.log 2>&1 &
        else
          appium --base-path=$APPIUM_BASE_PATH --log-timestamp \
            --log-no-colors --relaxed-security --default-capabilities \
            "{\"appium:deviceName\": \"$DEVICEFARM_DEVICE_NAME\", \
            \"platformName\": \"$DEVICEFARM_DEVICE_PLATFORM_NAME\", \
            \"appium:app\": \"$DEVICEFARM_APP_PATH\", \
            \"appium:udid\":\"$DEVICEFARM_DEVICE_UDID_FOR_APPIUM\", \
            \"appium:platformVersion\": \"$DEVICEFARM_DEVICE_OS_VERSION\", \
            \"appium:derivedDataPath\": \"$DEVICEFARM_WDA_DERIVED_DATA_PATH\", \
            \"appium:usePrebuiltWDA\": true, \
            \"appium:automationName\": \"XCUITest\"}" \
            >> $DEVICEFARM_LOG_DIR/appium.log 2>&1 &
        fi;

      # This code will wait until the Appium server starts.
      - |-
        appium_initialization_time=0;
        until curl --silent --fail "http://0.0.0.0:4723${APPIUM_BASE_PATH}/status"; do
          if [[ $appium_initialization_time -gt 30 ]]; then
            echo "Appium did not start within 30 seconds. Exiting...";
            exit 1;
          fi;
          appium_initialization_time=$((appium_initialization_time + 1));
          echo "Waiting for Appium to start on port 4723...";
          sleep 1;
        done;

  # The test phase contains commands for running your tests.
  test:
    commands:
      # Your test package is downloaded and unpackaged into the
$DEVICEFARM_TEST_PACKAGE_PATH directory.
```

```
      # When compiling with npm-bundle, the test folder can be found in the
 node_modules/*/ subdirectory.
      - cd $DEVICEFARM_TEST_PACKAGE_PATH/node_modules/*
      - echo "Starting the Appium NodeJS test"

      # Enter your command below to start the tests. The command should be the same
 command as the one
      # you use to run your tests locally from the command line. An example, "npm
 test", is given below:
      - npm test

  # The post-test phase contains commands that are run after your tests have completed.
  # If you need to run any commands to generating logs and reports on how your test
 performed,
  # we recommend adding them to this section.
  post_test:
    commands:

# Artifacts are a list of paths on the filesystem where you can store test output and
 reports.
# All files in these paths will be collected by Device Farm.
# These files will be available through the ListArtifacts API as your "Customer
 Artifacts".
artifacts:
  # By default, Device Farm will collect your artifacts from the $DEVICEFARM_LOG_DIR
 directory.
  - $DEVICEFARM_LOG_DIR
```

# Android 테스트를 위한 Amazon Linux 2 테스트 환경 작업

AWS Device Farm은 Amazon Linux 2를 실행하는 Amazon Elastic Compute Cloud(EC2) 호스트 머신을 활용하여 Android 테스트를 실행합니다. 테스트 실행을 예약하면 Device Farm은 각 디바이스에 전용 호스트를 할당하여 독립적으로 테스트를 실행합니다. 테스트 실행 후 생성된 아티팩트와 함께 호스트 컴퓨터가 종료됩니다.

Amazon Linux 2 테스트 호스트는 이전 Ubuntu 기반 시스템을 대체하는 최신 Android 테스트 환경입니다. 테스트 사양 파일을 사용하여 Amazon Linux 2 환경에서 Android 테스트를 실행하도록 선택할 수 있습니다.

Amazon Linux 2 호스트는 다음과 같은 장점이 있습니다.

- 더 빠르고 안정적인 테스트: 레거시 호스트에 비해 새 테스트 호스트는 테스트 속도를 크게 향상시키며 특히 테스트 시작 시간을 단축합니다. Amazon Linux 2 호스트는 테스트 중에 더 뛰어난 안정성과 신뢰성 역시 보여줍니다.
- 수동 테스트를 위한 향상된 원격 액세스: 최신 테스트 호스트로 업그레이드하면 Android 수동 테스트의 지연 시간이 줄어들고 비디오 성능이 향상됩니다.
- 표준 소프트웨어 버전 선택: Device Farm은 이제 Appium 프레임워크 버전 뿐만 아니라 테스트 호스트의 주요 프로그래밍 언어 지원을 표준화합니다. 지원되는 언어(현재 Java, Python, Node.js, Ruby) 및 Appium의 경우 새 테스트 호스트는 출시 직후 장기간 안정적인 릴리스를 제공합니다. `devicefarm-cli` 툴을 이용한 중앙 집중식 버전 관리로 프레임워크 전반에서 일관되게 테스트 사양 파일을 개발할 수 있습니다.

주제



## 지원 소프트웨어

Amazon Linux 2 테스트 호스트에는 Device Farm 테스트 프레임워크를 지원하는 여러 소프트웨어 라이브러리가 사전 설치되어 있어 시작(launch) 후 바로 사용할 수 있는 테스트 환경을 제공합니다. 기타 필수 소프트웨어의 경우 테스트 패키지에서 설치, 인터넷에서 다운로드, VPC 내 프라이빗 소스에 액세스하도록 테스트 사양 파일을 수정할 수 있습니다(자세한 내용은 VPC ENI 참조). 자세한 내용은 테스트 사양 파일 예제를 참조하세요.

현재 호스트에서 사용할 수 있는 소프트웨어 버전은 다음과 같습니다.

| 소프트웨어 라이브러리 | 소프트웨어 버전 | 테스트 사양 파일에 사용할 명령 |
| --- | --- | --- |
| Python | 3.8 | `devicefarm-cli use python 3.8` |

| | | |
|---|---|---|
| | 3.9 | `devicefarm-cli use python 3.9` |
| | 3.10 | `devicefarm-cli use python 3.10` |
| Java | 8 | `devicefarm-cli use java 8` |
| | 11 | `devicefarm-cli use java 11` |
| | 17 | `devicefarm-cli use java 17` |
| NodeJS | 16 | `devicefarm-cli use node 16` |
| | 18 | `devicefarm-cli use node 18` |
| Ruby | 2.7 | `devicefarm-cli use ruby 2.7` |
| | 3.2 | `devicefarm-cli use ruby 3.2` |
| Appium | 1 | `devicefarm-cli use appium 1` |
| | 2 | `devicefarm-cli use appium 2` |

테스트 호스트에는 `pip` 및 `npm` 패키지 관리자(Python과 Node.js 각각 포함)와 같은 개별 소프트웨어 버전에 일반적으로 지원되는 툴과 Appium과 같은 툴을 위한 의존성(예: Appium UIAutomator2 Driver) 역시 포함합니다. 따라서 지원되는 테스트 프레임워크를 사용하는 데 필요한 도구가 보장됩니다.

# devicefarm-cli 도구

Amazon Linux 2 테스트 호스트는 소프트웨어 버전을 선택하기 위해 devicefarm-cli로 불리는 표준화된 버전 관리 도구를 사용합니다. 이 도구는 Device Farm 테스트 AWS CLI 호스트와 별개이며 Device Farm 테스트 호스트에서만 사용할 수 있습니다. devicefarm-cli를 사용하면 테스트 호스트에 사전 설치된 소프트웨어 버전으로 전환할 수 있습니다. 이를 통해 시간이 지나도 Device Farm 테스트 사양 파일을 관리할 수 있는 명확한 방법을 제공하고 예측 가능한 메커니즘을 주어 향후 소프트웨어 버전을 업그레이드할 수 있도록 합니다.

아래 스니펫은 devicefarm-cli의 help 페이지를 보여줍니다.

```
$ devicefarm-cli help
 Usage: devicefarm-cli COMMAND [ARGS]

     Commands:
         help                          Prints this usage message.
         list                          Lists all versions of software configurable
                                       via this CLI.
         use <software> <version>      Configures the software for usage within the
                                       current shell's environment.
```

devicefarm-cli를 사용하여 몇 가지 예를 살펴봅시다. 이 도구를 사용하여 테스트 사양 파일에서 Python 버전을 *3.10*에서 *3.9*로 변경하려면 다음 명령을 실행하세요.

```
$ python --version
Python 3.10.12
$ devicefarm-cli use python 3.9
$ python --version
Python 3.9.17
```

Appium 버전을 *1*에서 *2*로 변경하려면 다음과 같습니다.

```
$ appium --version
1.22.3
$ devicefarm-cli use appium 2
$ appium --version
2.1.2
```

> **Tip**
>
> 소프트웨어 버전을 선택하면 Python을 위한 `pip`, NodeJS를 위한 `npm`과 같이 `devicefarm-cli` 또한 해당 언어에 대한 지원 도구로 전환한다는 점을 유의하세요.

## Android 테스트 호스트 선택

> **Warning**
>
> 기존 Android 테스트 호스트는 2024년 10월 21일에 더 이상 사용할 수 없습니다. 단, 지원 중단 절차는 여러 날짜로 나뉘어 진행됩니다.
>
> - 2024년 4월 22일에 모든 새 계정의 작업은 업그레이드된 테스트 호스트로 전달됩니다.
> - 2024년 9월 2일부터 모든 신규 또는 수정된 테스트 사양 파일은 업그레이드된 테스트 호스트를 대상으로 해야 합니다.
> - 2024년 10월 21일부터 더 이상 레거시 테스트 호스트에서 작업을 실행할 수 없게 됩니다.
>
> 호환성 문제를 방지하려면 테스트 사양 파일을 `amazon_linux_2` 호스트에 설정하십시오.

Android 테스트의 경우 Device Farm은 테스트 사양 파일 내 다음 필드에서 Amazon Linux 2 테스트 호스트 선택을 요구합니다.

```
android_test_host: amazon_linux_2 | legacy
```

Amazon Linux 2 테스트 호스트에서 테스트를 실행하기 위해 `amazon_linux_2`를 사용합니다.

```
android_test_host: amazon_linux_2
```

여기에서 Amazon Linux 2의 장점을 자세히 알아보세요.

Device Farm은 Android 테스트를 위해 레거시 호스트 환경 대신 Amazon Linux 2 호스트 사용을 권장합니다. 레거시 테스트 호스트 환경을 선호한다면 `legacy`를 사용하여 레거시 테스트 호스트에서 테스트를 실행하세요.

```
android_test_host: legacy
```

테스트 호스트를 선택하지 않은 테스트 사양 파일은 자동적으로 레거시 테스트 호스트에서 실행됩니다.

### 사용되지 않는 구문

다음은 테스트 사양 파일에서 Amazon Linux 2를 선택할 때 더 이상 사용되지 않는 구문입니다.

```
preview_features:
  android_amazon_linux_2_host: true
```

이 플래그를 사용하는 경우 Amazon Linux 2에서 테스트를 계속 실행합니다. 그러나 향후 유지 관리 오버헤드가 발생하지 않도록 `preview_features` 플래그 섹션을 제거하고 새 `android_test_host` 필드로 교체하는 것을 강력히 권장합니다.

> **Warning**
>
> 테스트 사양 파일에서 `android_test_host` 및 `android_amazon_linux_2_host` 플래그를 모두 사용하면 오류가 반환됩니다. 하나만 사용해야 하므로 `android_test_host`를 권장합니다.

## 테스트 사양 예입니다.

다음 스니펫은 Android용 Amazon Linux 2 테스트 호스트를 사용하여 Appium NodeJS 테스트 실행을 구성하는 Device Farm 테스트 사양 파일의 예입니다.

```
version: 0.1

# This flag enables your test to run using Device Farm's Amazon Linux 2 test host. For
 more information,
# please see https://docs.aws.amazon.com/devicefarm/latest/developerguide/amazon-
linux-2.html
android_test_host: amazon_linux_2

# Phases represent collections of commands that are executed during your test run on
 the test host.
phases:

  # The install phase contains commands for installing dependencies to run your tests.
  # For your convenience, certain dependencies are preinstalled on the test host. To
 lean about which
```

```
  # software is included with the host, and how to install additional software, please
 see:
  # https://docs.aws.amazon.com/devicefarm/latest/developerguide/amazon-linux-2-
supported-software.html

  # Many software libraries you may need are available from the test host using the
 devicefarm-cli tool.
  # To learn more about what software is available from it and how to use it, please
 see:
  # https://docs.aws.amazon.com/devicefarm/latest/developerguide/amazon-linux-2-
devicefarm-cli.html

  install:
    commands:
      # The Appium server is written using Node.js. In order to run your desired
 version of Appium,
      # you first need to set up a Node.js environment that is compatible with your
 version of Appium.
      - devicefarm-cli use node 18
      - node --version

      # Use the devicefarm-cli to select a preinstalled major version of Appium.
      - devicefarm-cli use appium 2
      - appium --version

      # The Device Farm service automatically updates the preinstalled Appium versions
 over time to
      # incorporate the latest minor and patch versions for each major version. If you
 wish to
      # select a specific version of Appium, you can use NPM to install it.
      # - npm install -g appium@2.1.3

      # For Appium version 2, Device Farm automatically updates the preinstalled
 UIAutomator2 driver
      # over time to incorporate the latest minor and patch versions for its major
 version 2. If you
      # want to install a specific version of the driver, you can use the Appium
 extension CLI to
      # uninstall the existing UIAutomator2 driver and install your desired version:
      # - appium driver uninstall uiautomator2
      # - appium driver install uiautomator2@2.34.0

      # We recommend setting the Appium server's base path explicitly for accepting
 commands.
```

```
      - export APPIUM_BASE_PATH=/wd/hub

      # Install the NodeJS dependencies.
      - cd $DEVICEFARM_TEST_PACKAGE_PATH
      # First, install dependencies which were packaged with the test package using
 npm-bundle.
      - npm install *.tgz
      # Then, optionally, install any additional dependencies using npm install.
      # If you do run these commands, we strongly recommend that you include your
 package-lock.json
      # file with your test package so that the dependencies installed on Device Farm
 match
      # the dependencies you've installed locally.
      # - cd node_modules/*
      # - npm install

  # The pre-test phase contains commands for setting up your test environment.
  pre_test:
    commands:

      # Appium downloads Chromedriver using a feature that is considered insecure for
 multitenant
      # environments. This is not a problem for Device Farm because each test host is
 allocated
      # exclusively for one customer, then terminated entirely. For more information,
 please see
      # https://github.com/appium/appium/blob/master/packages/appium/docs/en/guides/
security.md

      # We recommend starting the Appium server process in the background using the
 command below.
      # The Appium server log will be written to the $DEVICEFARM_LOG_DIR directory.
      # The environment variables passed as capabilities to the server will be
 automatically assigned
      # during your test run based on your test's specific device.
      # For more information about which environment variables are set and how they're
 set, please see
      # https://docs.aws.amazon.com/devicefarm/latest/developerguide/custom-test-
environment-variables.html
      - |-
        appium --base-path=$APPIUM_BASE_PATH --log-timestamp \
          --log-no-colors --relaxed-security --default-capabilities \
          "{\"appium:deviceName\": \"$DEVICEFARM_DEVICE_NAME\", \
          \"platformName\": \"$DEVICEFARM_DEVICE_PLATFORM_NAME\", \
```

```
          \"appium:app\": \"$DEVICEFARM_APP_PATH\", \
          \"appium:udid\":\"$DEVICEFARM_DEVICE_UDID\", \
          \"appium:platformVersion\": \"$DEVICEFARM_DEVICE_OS_VERSION\", \
          \"appium:chromedriverExecutableDir\":
\"$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR\", \
          \"appium:automationName\": \"UiAutomator2\"}" \
          >> $DEVICEFARM_LOG_DIR/appium.log 2>&amp;1 &amp;

     # This code will wait until the Appium server starts.
     - |-
       appium_initialization_time=0;
       until curl --silent --fail "http://0.0.0.0:4723${APPIUM_BASE_PATH}/status"; do
         if [[ $appium_initialization_time -gt 30 ]]; then
           echo "Appium did not start within 30 seconds. Exiting...";
           exit 1;
         fi;
         appium_initialization_time=$((appium_initialization_time + 1));
         echo "Waiting for Appium to start on port 4723...";
         sleep 1;
       done;

 # The test phase contains commands for running your tests.
 test:
   commands:
     # Your test package is downloaded and unpackaged into the
$DEVICEFARM_TEST_PACKAGE_PATH directory.
     # When compiling with npm-bundle, the test folder can be found in the
node_modules/*/ subdirectory.
     - cd $DEVICEFARM_TEST_PACKAGE_PATH/node_modules/*
     - echo "Starting the Appium NodeJS test"

     # Enter your command below to start the tests. The command should be the same
command as the one
     # you use to run your tests locally from the command line. An example, "npm
test", is given below:
     - npm test

 # The post-test phase contains commands that are run after your tests have completed.
 # If you need to run any commands to generating logs and reports on how your test
performed,
 # we recommend adding them to this section.
 post_test:
   commands:
```

```
# Artifacts are a list of paths on the filesystem where you can store test output and
 reports.
# All files in these paths will be collected by Device Farm.
# These files will be available through the ListArtifacts API as your "Customer
 Artifacts".
artifacts:
  # By default, Device Farm will collect your artifacts from the $DEVICEFARM_LOG_DIR
 directory.
  - $DEVICEFARM_LOG_DIR
```

## Amazon Linux 2 테스트 호스트로 마이그레이션

> **Warning**
>
> 기존 Android 테스트 호스트는 2024년 10월 21일에 더 이상 사용할 수 없습니다. 단, 지원 중단 절차는 여러 날짜로 나뉘어 진행됩니다.
>
> - 2024년 4월 22일에 모든 새 계정의 작업은 업그레이드된 테스트 호스트로 전달됩니다.
> - 2024년 9월 2일부터 모든 신규 또는 수정된 테스트 사양 파일은 업그레이드된 테스트 호스트를 대상으로 해야 합니다.
> - 2024년 10월 21일부터 더 이상 레거시 테스트 호스트에서 작업을 실행할 수 없게 됩니다.
>
> 호환성 문제를 방지하려면 테스트 사양 파일을 `amazon_linux_2` 호스트에 설정하십시오.

기존 테스트를 레거시 호스트에서 새 Amazon Linux 2 호스트로 마이그레이션하려면 기존 테스트를 기반으로 새 테스트 사양 파일을 개발하세요. 테스트 유형에 맞는 새 기본 테스트 사양 파일로 시작하는 것을 권장합니다. 다음으로 기존 테스트 사양 파일에서 새 파일로 관련 명령을 마이그레이션하고 이전 파일을 백업으로 저장합니다. 이를 통해 기존 코드를 재사용하면서 새 호스트에 최적화된 기본 사양을 활용할 수 있습니다. 따라서 테스트에 최적으로 구성된 새 호스트의 이점을 최대한 활용하는 동시에 기존 레거시 테스트 사양을 참조용으로 유지하여 새 환경에 맞게 명령을 적용할 수 있습니다.

이전 테스트 사양 파일의 명령을 재 사용하여 새 Amazon Linux 2 테스트 사양 파일을 생성하는 단계는 다음과 같습니다.

1. https://console.aws.amazon.com/devicefarm에서 Device Farm 콘솔에 로그인하세요.
2. 자동화 테스트가 포함된 Device Farm 프로젝트로 이동하세요.
3. 프로젝트에서 새 테스트 실행 만들기를 선택하세요.

4. 테스트 프레임워크를 위해 이전에 사용한 앱과 테스트 패키지를 선택하세요.
5. 사용자 지정 환경에서 테스트 실행을 선택하세요.
6. 테스트 사양 드롭다운 메뉴에서 레거시 테스트 호스트의 테스트에서 현재 사용 중인 테스트 사양 파일을 선택하세요.
7. 나중에 참조할 수 있도록 이 파일의 내용을 복사하여 텍스트 편집기에 로컬로 붙여넣습니다.
8. 테스트 사양 드롭다운 메뉴에서 테스트 사양 선택을 가장 최근의 기본 테스트 사양 파일로 변경하세요.
9. 편집을 선택하여 테스트 사양 편집 인터페이스로 들어가세요. 테스트 사양 파일 내 첫 몇 줄을 통해 새 테스트 호스트가 선택된 것을 확인하세요.

```
android_test_host: amazon_linux_2
```

10. 여기에서 테스트 호스트를 선택하기 위한 구문을 검토하고 여기에서 테스트 호스트 간의 주요 차이점을 확인하세요.
11. 6단계에서 로컬에 저장한 테스트 사양 파일의 명령을 새 기본 테스트 사양 파일에 선택적으로 추가하고 편집하세요. 그런 다음 다른 이름으로 저장을 선택하여 새 사양 파일을 저장하세요. 이제 Amazon Linux 2 테스트 호스트에서 테스트 실행을 예약할 수 있습니다.

## 새 테스트 호스트와 레거시 테스트 호스트 간의 차이점

Amazon Linux 2 테스트 호스트 사용을 위해 테스트 사양 파일을 편집하고 레거시 테스트 호스트에서 테스트를 전환하려면 다음과 같은 주요 환경 차이에 유의하세요.

- 소프트웨어 버전 선택: 대부분의 경우, 기본 소프트웨어 버전이 변경되었으므로 이전에 레거시 테스트 호스트에서 소프트웨어 버전을 명시적으로 선택하지 않았다면 `devicefarm-cli`를 사용하여 Amazon Linux 2 테스트 호스트에서 지정하는 것이 좋습니다. 대부분의 사용 사례에서 고객이 사용하는 소프트웨어 버전을 명시적으로 선택하는 것을 권장합니다. `devicefarm-cli`를 통해 소프트웨어 버전을 선택하면 예측 가능하고 일관된 경험이 가능하며 Device Farm이 테스트 호스트에서 해당 버전을 제거하려는 경우 충분한 경고를 받습니다.

  또한, 새로운 `devicefarm-cli` 소프트웨어 선택 시스템을 위해 `nvm`, `pyenv`, `avm`, `rvm`과 같은 소프트웨어 선택 도구가 제거되었습니다.
- 사용 가능한 소프트웨어 버전: 기존에 사전 설치된 소프트웨어의 여러 버전이 제거되었으며 많은 새로운 버전이 추가되었습니다. 따라서 `devicefarm-cli`를 사용하여 소프트웨어 버전을 선택할 때는 지원되는 버전 목록에 있는 버전을 선택해야 합니다.

- 레거시 호스트 테스트 사양 파일에 절대 경로로 하드 코딩된 파일 경로는 Amazon Linux 2 테스트 호스트에서 예상대로 작동하지 않을 가능성이 높으므로 일반적으로 테스트 사양 파일 사용에 권장되지 않습니다. 모든 테스트 사양 파일 코드에 상대 경로와 환경 변수를 사용하는 것이 좋습니다. 또한 테스트에 필요한 대부분의 바이너리는 호스트의 PATH에서 찾을 수 있으므로 사양 파일에서 이름을 사용하는 것만으로(예: appium) 즉시 실행할 수 있습니다.
- 현재 새 테스트 호스트에서는 성능 데이터 수집이 지원되지 않습니다.
- 운영 체제 버전: 레거시 테스트 호스트는 Ubuntu 운영 체제를 기반으로 했지만 새 호스트는 Amazon Linux 2를 기반으로 합니다. 따라서 사용 가능한 시스템 라이브러리와 시스템 라이브러리 버전에서 약간의 사용 차이를 느낄 수 있습니다.
- Appium Java를 사용하는 경우 이전 호스트는 TestNG 프레임워크(환경 변수 `$DEVICEFARM_TESTNG_JAR`를 통한) 파일을 포함한 반면, 새 테스트 호스트의 클래스 경로에 사전 설치된 JAR 파일이 없습니다. 테스트 프레임워크에 필요한 JAR 파일을 테스트 패키지 내에 패키징하고 테스트 사양 파일에서 `$DEVICEFARM_TESTNG_JAR` 변수 인스턴스를 제거하는 것이 좋습니다. 자세한 내용은 Appium 및 AWS Device Farm 사용을 참조하세요.
- Appium을 사용하는 경우 고객이 Android용 Chromedriver에 액세스할 수 있도록 하는 새로운 접근 방식을 위해 `$DEVICEFARM_CHROMEDRIVER_EXECUTABLE` 환경 변수가 제거되었습니다. 새 환경 변수 `$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR`을 사용하는 예시로 기본 테스트 사양 파일을 참조하세요.

> **Note**
>
> 기존 Appium 서버 명령을 레거시 테스트 사양 파일에서 그대로 유지하는 것을 강력히 권장합니다.

소프트웨어 관점에서 테스트 호스트 간의 차이점에 대한 피드백이나 질문이 있는 경우 지원 사례를 통해 서비스 팀에 문의하는 것이 좋습니다.

## 환경 변수

환경 변수는 자동 테스트에서 사용하는 값을 나타냅니다. YAML 파일 및 테스트 코드에서 이러한 환경 변수를 사용할 수 있습니다. 사용자 지정 테스트 환경에서 Device Farm은 실행 시 환경 변수를 동적으로 채웁니다.

주제

- 공통 환경 변수

- Appium Java JUnit 환경 변수
- Appium Java TestNG 환경 변수
- XCUITest 환경 변수

# 공통 환경 변수

## Android 테스트

이 단원에서는 Device Farm에서 지원하는 Android 플랫폼 테스트에 공통적인 사용자 지정 환경 변수를 설명합니다.

**$DEVICEFARM_DEVICE_NAME**

테스트를 실행하는 디바이스의 이름 디바이스의 고유한 디바이스 식별자(UDID)

**$DEVICEFARM_DEVICE_PLATFORM_NAME**

디바이스 플랫폼 이름 Android 또는 iOS입니다.

**$DEVICEFARM_DEVICE_OS_VERSION**

디바이스 OS 버전

**$DEVICEFARM_APP_PATH**

테스트가 실행되고 있는 호스트 머신의 모바일 앱 경로 앱 경로는 모바일 앱에서만 사용할 수 있습니다.

**$DEVICEFARM_DEVICE_UDID**

자동 테스트를 실행 중인 모바일 디바이스의 고유한 식별자

**$DEVICEFARM_LOG_DIR**

테스트 실행 중에 생성되는 로그 파일의 경로 기본적으로 이 디렉터리의 모든 파일은 ZIP 파일에 보관되며 테스트 실행 후 아티팩트로 사용할 수 있습니다.

**$DEVICEFARM_SCREENSHOT_PATH**

테스트 실행 중에 캡처되는 스크린샷이 있는 경우 경로

**$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR**

Appium 웹 및 하이브리드 테스트에서 사용하는 데 필요한 Chromedriver 실행 파일이 들어 있는 디렉터리의 위치

**$ANDROID_HOME**

Android SDK 설치 디렉터리의 경로

> **Note**
>
> ANDROID_HOME 환경 변수는 Android용 Amazon Linux 2 테스트 호스트에서만 사용할 수 있습니다.

### iOS 테스트

이 단원에서는 Device Farm에서 지원하는 iOS 플랫폼 테스트에 공통적인 사용자 지정 환경 변수를 설명합니다.

**$DEVICEFARM_DEVICE_NAME**

테스트를 실행하는 디바이스의 이름 디바이스의 고유한 디바이스 식별자(UDID)

**$DEVICEFARM_DEVICE_PLATFORM_NAME**

디바이스 플랫폼 이름 Android 또는 iOS입니다.

**$DEVICEFARM_APP_PATH**

테스트가 실행되고 있는 호스트 머신의 모바일 앱 경로 앱 경로는 모바일 앱에서만 사용할 수 있습니다.

**$DEVICEFARM_DEVICE_UDID**

자동 테스트를 실행 중인 모바일 디바이스의 고유한 식별자

**$DEVICEFARM_LOG_DIR**

테스트 실행 중에 생성되는 로그 파일의 경로

**$DEVICEFARM_SCREENSHOT_PATH**

테스트 실행 중에 캡처되는 스크린샷이 있는 경우의 경로입니다.

## Appium Java JUnit 환경 변수

이 단원에서는 사용자 지정 테스트 환경에서 Appium Java JUnit 테스트에서 사용되는 환경 변수를 설명합니다.

**$DEVICEFARM_TESTNG_JAR**

TestNG .jar 파일의 경로

**$DEVICEFARM_TEST_PACKAGE_PATH**

테스트 패키지 파일의 압축을 푼 콘텐츠의 경로

## Appium Java TestNG 환경 변수

이 단원에서는 사용자 지정 테스트 환경에서 Appium Java TestNG 테스트에서 사용되는 환경 변수를 설명합니다.

**$DEVICEFARM_TESTNG_JAR**

TestNG .jar 파일의 경로

**$DEVICEFARM_TEST_PACKAGE_PATH**

테스트 패키지 파일의 압축을 푼 콘텐츠의 경로

## XCUITest 환경 변수

**$DEVICEFARM_XCUITESTRUN_FILE**

Device Farm `.xctestun` 파일의 경로 앱 및 테스트 패키지에서 생성됩니다.

**$DEVICEFARM_DERIVED_DATA_PATH**

Device Farm xcodebuild 출력의 예상되는 경로

# 표준 테스트 환경에서 사용자 지정 테스트 환경으로 테스트 마이그레이션

다음 가이드에서는 표준 테스트 실행 모드에서 사용자 지정 실행 모드로 전환하는 방법을 설명합니다. 마이그레이션에는 주로 두 가지 다른 형태의 실행이 포함됩니다.

1. 표준 모드: 이 테스트 실행 모드는 주로 고객에게 세분화된 보고 및 완전히 관리되는 환경을 제공하기 위해 구축되었습니다.

2. 사용자 지정 모드: 이 테스트 실행 모드는 더 빠른 테스트 실행, 리프트 앤 시프트 기능, 로컬 환경과 동등한 수준 달성, 라이브 비디오 스트리밍이 필요한 다양한 사용 사례에 맞게 구축되었습니다.

## 마이그레이션 시 고려 사항

이 섹션에는 사용자 지정 모드로 마이그레이션할 때 고려해야 할 몇 가지 주요 사용 사례가 나열되어 있습니다.

1. 속도: 표준 실행 모드에서 Device Farm은 특정 프레임워크의 패키징 지침을 사용하여 패키징하고 업로드한 테스트의 메타데이터를 파싱합니다. 파싱은 패키지의 테스트 수를 감지합니다. 이후 Device Farm은 각 테스트를 개별적으로 실행하고 각 테스트에 대한 로그, 비디오 및 기타 결과 아티팩트를 개별적으로 표시합니다. 하지만 end-to-end 테스트 및 결과 아티팩트의 사전 및 사후 처리가 서비스 측에서 발생하므로 총 테스트 실행 시간이 꾸준히 늘어납니다.

   반대로 사용자 지정 실행 모드는 테스트 패키지를 파싱하지 않으므로 테스트 또는 결과 아티팩트에 대한 사전 처리가 필요 없고 사후 처리가 최소화됩니다. 그 결과 총 end-to-end 실행 시간이 로컬 설정과 비슷해집니다. 테스트는 로컬 시스템에서 실행할 때와 동일한 형식으로 실행됩니다. 테스트 결과는 로컬에서 얻은 결과와 동일하며 작업 실행 종료 시 다운로드할 수 있습니다.
2. 사용자 지정 또는 유연성: 표준 실행 모드는 테스트 패키지를 파싱하여 테스트 수를 감지한 다음 각 테스트를 개별적으로 실행합니다. 단, 테스트가 지정한 순서대로 실행된다는 보장은 없습니다. 따라서 특정 실행 시퀀스가 필요한 테스트는 예상대로 작동하지 않을 수 있습니다. 또한 호스트 컴퓨터 환경을 사용자 지정하거나 특정 방식으로 테스트를 실행하는 데 필요할 수 있는 구성 파일을 전달할 방법도 없습니다.

   반면 사용자 지정 모드에서는 추가 소프트웨어 설치, 테스트에 필터 전달, 구성 파일 전달, 테스트 실행 설정 제어 등의 기능을 비롯한 호스트 컴퓨터 환경을 구성할 수 있습니다. 이 작업은 YAML 파일(testspec 파일이라고도 함)을 통해 수행되며, 이 파일에 쉘 명령을 추가하여 수정할 수 있습니다. 이 YAML 파일은 테스트 호스트 시스템에서 실행되는 쉘 스크립트로 변환됩니다. 여러 YAML 파일을 저장하고 실행을 예약할 때 요구 사항에 따라 동적으로 하나를 선택할 수 있습니다.
3. 라이브 비디오 및 로깅: 표준 실행 모드와 사용자 지정 실행 모드 모두 테스트에 필요한 비디오와 로그를 제공합니다. 하지만 표준 모드에서는 테스트가 완료된 후에만 테스트의 비디오 및 사전 정의된 로그를 얻을 수 있습니다.

   반면 사용자 지정 모드에서는 비디오의 실시간 스트림과 테스트의 클라이언트 측 로그를 제공합니다. 또한 테스트 종료 시 비디오 및 기타 아티팩트를 다운로드할 수 있습니다.
4. 사용 중단: 표준 실행 모드에서는 다음 테스트 유형이 2023년 12월 말까지 지원 중단될 예정입니다.
   - Appium(모든 언어)

- Calabash
- XCTest
- UI 자동화
- UI 오토메이터
- 웹 테스트
- 기본 제공 탐색기

지원 중단되면 표준 모드에서는 이러한 프레임워크를 사용할 수 없습니다. 대신 위에 나열된 테스트 유형에 대해 사용자 지정 모드를 사용할 수 있습니다.

> **Tip**
>
> 사용 사례에 위의 요인 중 하나 이상이 포함되는 경우 사용자 지정 실행 모드로 전환하는 것이 좋습니다.

## 마이그레이션 단계

표준 모드에서 사용자 지정 모드로 마이그레이션하려면 다음을 수행합니다.

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/codedeploy/에서 Device Farm 콘솔을 엽니다.
2. 프로젝트를 선택한 다음 새 자동화 실행을 시작하세요.
3. 앱을 업로드(또는 `web app`을 선택)하고 테스트 프레임워크 유형을 선정하여 테스트 패키지를 업로드한 다음, `Choose your execution environment` 파라미터 아래에서 옵션을 `Run your test in a custom environment`로 선택합니다.
4. 기본적으로 Device Farm의 예제 테스트 사양 파일이 표시되어 보고 편집할 수 있습니다. 이 예제 파일은 사용자 지정 환경 모드에서 테스트를 시험해 보기 위한 출발점으로 사용할 수 있습니다. 그런 다음 콘솔에서 테스트가 제대로 작동하는지 확인한 후에는 Device Farm과의 API, CLI 및 파이프라인 통합을 변경하여 테스트 실행을 예약할 때 이 테스트 사양 파일을 파라미터로 사용할 수 있습니다. 테스트 사양 파일을 실행용 파라미터로 추가하는 방법에 대한 자세한 내용은 API 가이드의 `ScheduleRun` API용 `testSpecArn` 파라미터 섹션을 참조하세요.

## Appium 프레임워크

사용자 지정 테스트 환경에서 Device Farm은 Appium 프레임워크 테스트에 Appium 기능을 삽입하거나 재정의하지 않습니다. 테스트 사양 YAML 파일 또는 테스트 코드에서 테스트의 Appium 기능을 지정해야 합니다.

## Android 계측

Android 계측 테스트를 사용자 지정 테스트 환경으로 이동할 때 변경이 필요하지 않습니다.

## iOS XCUITest

iOS XCUITest 테스트를 사용자 지정 테스트 환경으로 이동할 때 변경이 필요하지 않습니다.

# Device Farm의 사용자 지정 테스트 환경 확장

Device Farm 사용자 지정 모드를 사용하면 테스트 스위트 외에도 다양한 기능을 실행할 수 있습니다. 이 단원에서는 테스트 스위트를 확장하고 테스트를 최적화하는 방법을 알아봅니다.

## PIN 설정

일부 응용 프로그램에서는 장치에 PIN을 설정해야 합니다. Device Farm은 기본적으로 장치에 PIN 설정을 지원하지 않습니다. 하지만 다음과 같은 경고 사항과 함께 가능합니다.

- 디바이스는 Android 8 이상을 실행해야 합니다.
- 테스트가 완료된 후 PIN을 제거해야 합니다.

테스트에서 PIN을 설정하려면 다음과 같이 `pre_test` 및 `post_test` 단계를 사용하여 PIN을 설정하고 제거합니다.

```
phases:
    pre_test:
      - # ... among your pre_test commands
      - DEVICE_PIN_CODE="1234"
      - adb shell locksettings set-pin "$DEVICE_PIN_CODE"
    post_test:
      - # ... Among your post_test commands
      - adb shell locksettings clear --old "$DEVICE_PIN_CODE"
```

테스트 스위트가 시작되면 PIN 1234가 설정됩니다. 테스트 스위트가 종료되면 PIN이 제거됩니다.

> **Warning**
>
> 테스트가 완료된 후 디바이스에서 PIN을 제거하지 않으면 디바이스와 계정이 격리됩니다.

## 원하는 기능을 통해 Appium 기반 테스트 속도 향상

Appium을 사용할 때 표준 모드 테스트 스위트가 매우 느릴 수 있습니다. 이는 Device Farm이 기본 설정을 적용하고 Appium 환경 사용 방법에 대해 어떠한 가정도 하지 않기 때문입니다. 이러한 기본값은 업계 모범 사례를 기반으로 구축되었지만 사용 상황에 따라 적용되지 않을 수 있습니다. Appium 서버의 파라미터를 미세 조정하려면 테스트 사양에서 기본 Appium 기능을 조정할 수 있습니다. 예를 들어 다음은 iOS 테스트 스위트에서 초기 시작 시간을 단축하기 위해 `usePrebuildWDA` 기능을 `true`로 설정합니다.

```
phases:
  pre_test:
    - # ... Start up Appium
    - >-
    appium --log-timestamp
    --default-capabilities "{\"usePrebuiltWDA\": true, \"derivedDataPath\":
\"$DEVICEFARM_WDA_DERIVED_DATA_PATH\",
    \"deviceName\": \"$DEVICEFARM_DEVICE_NAME\", \"platformName\":
\"$DEVICEFARM_DEVICE_PLATFORM_NAME\", \"app\":\"$DEVICEFARM_APP_PATH\",
    \"automationName\":\"XCUITest\", \"udid\":\"$DEVICEFARM_DEVICE_UDID_FOR_APPIUM\",
 \"platformVersion\":\"$DEVICEFARM_DEVICE_OS_VERSION\"}"
    >> $DEVICEFARM_LOG_DIR/appiumlog.txt 2>&1 &
```

Appium 기능은 쉘로 이스케이프 처리되고 인용된 JSON 구조여야 합니다.

다음과 같은 Appium 기능은 성능 향상의 일반적인 출처입니다.

`noReset` 및 `fullReset`

상호 배타적인 이 두 기능은 각 세션이 완료된 후 Appium의 동작을 설명합니다. `noReset`을 `true`로 설정하면 Appium 세션이 종료될 때 Appium 서버가 애플리케이션에서 데이터를 제거하지

않으므로 사실상 어떤 정리 작업도 수행하지 않습니다. `fullReset`은 세션이 종료된 후 장치에서 모든 애플리케이션 데이터를 제거하고 지웁니다. 자세한 내용은 Appium 설명서의 재설정 전략을 참조하세요.

`ignoreUnimportantViews`(Android만 해당)

Appium에 Android UI 계층 구조를 테스트와 관련된 뷰로만 압축하여 특정 요소 조회 속도를 높이도록 지시합니다. 하지만 이렇게 하면 UI 레이아웃의 계층 구조가 변경되어 일부 XPath 기반 테스트 스위트가 손상될 수 있습니다.

`skipUnlock`(Android만 해당)

Appium에 현재 설정된 PIN 코드가 없음을 알립니다. 이렇게 하면 화면 끄기 이벤트 또는 기타 잠금 이벤트 후 테스트 속도가 빨라집니다.

`webDriverAgentUrl`(iOS만 해당)

`webDriverAgent`인 필수 iOS 종속 항목이 이미 실행 중이고 지정된 URL에서 HTTP 요청을 수락할 수 있다고 가정하도록 Appium에 지시합니다. 아직 `webDriverAgent`가 실행되지 않은 경우 Appium의 테스트 스위트 초기에 `webDriverAgent`를 시작하는 데 시간이 걸릴 수 있습니다. Appium을 시작할 때 `webDriverAgent`를 직접 시작하고 `webDriverAgentUrl`을 `http://localhost:8100`로 설정하면 테스트 스위트를 더 빠르게 부팅할 수 있습니다. 참고로 이 기능을 `useNewWDA` 기능과 함께 사용해서는 안 됩니다.

다음 코드를 사용하여 디바이스의 로컬 포트 `8100`에 있는 테스트 사양 파일에서 `webDriverAgent`를 시작한 다음 테스트 호스트의 로컬 포트 `8100`로 전달할 수 있습니다(이렇게 하면 `webDriverAgentUrl` 값을 `http://localhost:8100`로 설정할 수 있음). 이 코드는 Appium 및 `webDriverAgent` 환경 변수를 설정하기 위한 코드를 정의한 후 설치 단계에서 실행해야 합니다.

```
      # Start WebDriverAgent and iProxy
      - >-
        xcodebuild test-without-building -project /usr/local/avm/versions/
$APPIUM_VERSION/node_modules/appium/node_modules/appium-webdriveragent/
WebDriverAgent.xcodeproj
        -scheme WebDriverAgentRunner -derivedDataPath
 $DEVICEFARM_WDA_DERIVED_DATA_PATH
        -destination id=$DEVICEFARM_DEVICE_UDID_FOR_APPIUM
 IPHONEOS_DEPLOYMENT_TARGET=$DEVICEFARM_DEVICE_OS_VERSION
        GCC_TREAT_WARNINGS_AS_ERRORS=0 COMPILER_INDEX_STORE_ENABLE=NO >>
 $DEVICEFARM_LOG_DIR/webdriveragent_log.txt 2>&1 &
```

```
        iproxy 8100 8100 >> $DEVICEFARM_LOG_DIR/iproxy_log.txt 2>&1 &
```

그런 다음 테스트 사양 파일에 다음 코드를 추가하여 webDriverAgent가 성공적으로 시작되었는지 확인할 수 있습니다. Appium이 성공적으로 시작되었는지 확인한 후 사전 테스트 단계가 끝날 때 이 코드를 실행해야 합니다.

```
      # Wait for WebDriverAgent to start
      - >-
        start_wda_timeout=0;
        while [ true ];
        do
          if [ $start_wda_timeout -gt 60 ];
          then
              echo "WebDriverAgent server never started in 60 seconds.";
              exit 1;
          fi;
          grep -i "ServerURLHere" $DEVICEFARM_LOG_DIR/webdriveragent_log.txt >> /
dev/null 2>&1;
          if [ $? -eq 0 ];
          then
              echo "WebDriverAgent REST http interface listener started";
              break;
          else
              echo "Waiting for WebDriverAgent server to start. Sleeping for 1
 seconds";
              sleep 1;
              start_wda_timeout=$((start_wda_timeout+1));
          fi;
        done;
```

Appium이 지원하는 기능에 대한 자세한 내용은 Appium 설명서의 Appium 필요한 기능을 참조하세요.

## 테스트 실행 후 Webhooks 및 기타 API 사용

모든 테스트 스위트의 curl 사용이 완료된 후 Device Farm이 webhook을 호출하도록 할 수 있습니다. 이 작업을 수행하는 프로세스는 대상 및 형식에 따라 다릅니다. 특정 webhook에 대해서는 해당 Webhook의 설명서를 참조하세요. 다음 예제는 테스트 스위트가 완료될 때마다 Slack webhook에 메시지를 게시합니다.

```
phases:
  post_test:
```

```
    - curl -X POST -H 'Content-type: application/json' --data '{"text":"Tests on
 '$DEVICEFARM_DEVICE_NAME' have finished!"}' https://hooks.slack.com/services/
T00000000/B00000000/XXXXXXXXXXXXXXXXXXXXXXX
```

Slack에서 webhook를 사용하는 방법에 대한 자세한 내용은 Slack API 참조의 Webhook을 사용하여 첫 번째 Slack 메시지 보내기를 참조하세요.

Webhook을 호출하는 용도로만 curl을 사용할 수 있는 것은 아닙니다. 테스트 패키지는 Device Farm 실행 환경과 호환되는 한 추가 스크립트 및 도구를 포함할 수 있습니다. 예를 들어 테스트 패키지에는 다른 API에 요청을 하는 보조 스크립트가 포함될 수 있습니다. 모든 필수 패키지가 테스트 스위트의 요구 사항과 함께 설치되어 있는지 확인하세요. 테스트 스위트가 완성된 후 실행되는 스크립트를 추가하려면 테스트 패키지에 스크립트를 포함하고 테스트 사양에 다음을 추가하세요.

```
phases:
  post_test:
    - python post_test.py
```

> **Note**
>
> 테스트 패키지에 사용된 API 키 또는 기타 인증 토큰을 유지 관리하는 것은 사용자의 책임입니다. 모든 형태의 보안 인증 정보를 소스 제어에서 제외시키고, 권한이 가장 적은 자격 증명을 사용하며, 가능하면 취소 가능하고 수명이 짧은 토큰을 사용하는 것이 좋습니다. 보안 요구 사항을 확인하려면 사용하는 타사 API의 설명서를 참조하세요.

AWS 서비스를 테스트 실행 스위트의 일부로 사용할 계획이라면 테스트 스위트 외부에서 생성되고 테스트 패키지에 포함된 IAM 임시 자격 증명을 사용해야 합니다. 이러한 자격 증명은 부여된 권한이 가장 적고 수명이 가장 짧아야 합니다. 임시 보안 인증 생성에 대한 자세한 내용은 IAM 사용 설명서의 임시 보안 인증 요청 단원을 참조하세요.

## 테스트 패키지에 추가 파일 추가

추가 파일을 추가 구성 파일 또는 추가 테스트 데이터인 테스트 파일의 일부로 사용할 수 있습니다. AWS Device Farm에 업로드하기 전에 테스트 패키지에 이러한 추가 파일을 추가한 다음 사용자 지정 환경 모드에서 액세스할 수 있습니다. 기본적으로 모든 테스트 패키지 업로드 형식(ZIP, IPA, APK, JAR 등)은 표준 ZIP 작업을 지원하는 패키지 아카이브 형식입니다.

업로드하기 전에 다음 명령어를 사용하여 테스트 아카이브에 AWS Device Farm 파일을 추가할 수 있습니다.

```
$ zip zip-with-dependencies.zip extra_file
```

추가 파일이 있는 디렉터리의 경우

```
$ zip -r zip-with-dependencies.zip extra_files/
```

이 명령은 IPA 파일을 제외한 모든 테스트 패키지 업로드 형식에서 예상대로 작동합니다. IPA 파일의 경우, 특히 XCuiTests와 함께 사용하는 경우 iOS 테스트 패키지를 AWS Device Farm 재설계하는 방식 때문에 추가 파일은 약간 다른 위치에 두는 것이 좋습니다. iOS 테스트를 빌드할 때 테스트 애플리케이션 디렉터리는 *Payload*라는 다른 디렉터리 내에 위치합니다.

예를 들어, iOS 테스트 디렉터리 중 하나는 다음과 같습니다.

```
$ tree
.
### Payload
    ### ADFiOSReferenceAppUITests-Runner.app
        ### ADFiOSReferenceAppUITests-Runner
        ### Frameworks
        #   ### XCTAutomationSupport.framework
        #   #   ### Info.plist
        #   #   ### XCTAutomationSupport
        #   #   ### _CodeSignature
        #   #   #   ### CodeResources
        #   #   ### version.plist
        #   ### XCTest.framework
        #       ### Info.plist
        #       ### XCTest
        #       ### _CodeSignature
        #       #   ### CodeResources
        #       ### en.lproj
        #       #   ### InfoPlist.strings
        #       ### version.plist
        ### Info.plist
        ### PkgInfo
        ### PlugIns
        #   ### ADFiOSReferenceAppUITests.xctest
        #   #   ### ADFiOSReferenceAppUITests
        #   #   ### Info.plist
        #   #   ### _CodeSignature
        #   #       ### CodeResources
        #   ### ADFiOSReferenceAppUITests.xctest.dSYM
```

```
        #       ### Contents
        #           ### Info.plist
        #           ### Resources
        #               ### DWARF
        #                   ### ADFiOSReferenceAppUITests
        ### _CodeSignature
        #   ### CodeResources
        ### embedded.mobileprovision
```

이러한 XCUITest 패키지의 경우 *Payload* 디렉터리 안의 *.app*으로 끝나는 디렉터리에 추가 파일을 추가합니다. 예를 들어, 다음 명령은 이 테스트 패키지에 파일을 추가하는 방법을 보여줍니다.

```
$ mv extra_file Payload/*.app/
$ zip -r my_xcui_tests.ipa Payload/
```

테스트 패키지에 파일을 추가하면 업로드 형식에 따라 AWS Device Farm 에서 상호 작용 동작이 약간 다를 수 있습니다. 업로드에서 ZIP 파일 확장자를 사용한 경우 AWS Device Farm 이 테스트 전에 업로드의 압축을 자동으로 풀고 *$DEVICEFARM_TEST_PACKAGE_PATH* 환경 변수가 있는 위치에 압축을 푼 파일을 둡니다. (즉, 첫 번째 예제에서처럼 아카이브의 루트에 *extra_file*이라는 파일을 추가하면 테스트 중에 해당 파일은 *$DEVICEFARM_TEST_PACKAGE_PATH/extra_file*에 위치하게 됩니다).

좀 더 실용적인 예제를 사용하려면 테스트에 *testng.xml* 파일을 포함하려는 Appium TestNG 사용자인 경우 다음 명령을 사용하여 아카이브에 포함시킬 수 있습니다.

```
$ zip zip-with-dependencies.zip testng.xml
```

그런 다음 사용자 지정 환경 모드에서 테스트 명령을 다음과 같이 변경할 수 있습니다.

```
java -D appium.screenshots.dir=$DEVICEFARM_SCREENSHOT_PATH org.testng.TestNG -testjar
 *-tests.jar -d $DEVICEFARM_LOG_DIR/test-output $DEVICEFARM_TEST_PACKAGE_PATH/
testng.xml
```

테스트 패키지 업로드 확장자가 ZIP이 아닌 경우(예: APK, IPA 또는 JAR 파일) 업로드된 패키지 파일 자체는 *$DEVICEFARM_TEST_PACKAGE_PATH*에서 찾을 수 있습니다. 이러한 파일은 여전히 아카이브 형식 파일이므로 파일 압축을 풀어 내에서 추가 파일에 액세스할 수 있습니다. 예를 들어 다음 명령은 테스트 패키지의 콘텐츠(APK, IPA 또는 JAR 파일용)를 */tmp* 디렉터리에 압축 해제합니다.

```
unzip $DEVICEFARM_TEST_PACKAGE_PATH -d /tmp
```

APK 또는 JAR 파일의 경우 */tmp ####*(예: */tmp/extra_file*)에 추가 파일이 압축 해제되어 있는 것을 확인할 수 있습니다. IPA 파일의 경우 앞서 설명한 것처럼 *Payload* 디렉터리 내에 있는 *.app*으로 끝나는 폴더 내에서는 추가 파일이 약간 다른 위치에 있게 됩니다. 예를 들어 위의 IPA 예제를 기반으로 하면 파일은 */TMP/####/ADFIOS ReferenceApp UITESTS-Runner.app/extra_file* (*/tmp/payload/ *.app/extra_file#* 참조 가능) 위치에서 찾을 수 있습니다.

# AWS Device Farm에서의 원격 액세스 작업

원격 액세스 기능을 사용하면 기능을 테스트하고 고객 문제를 재현하기 위해 웹 브라우저를 통해 실시간으로 살짝 밀기와 제스처를 수행하고 디바이스와 상호 작용할 수 있습니다. 해당 디바이스와 원격 액세스 세션을 만들어 특정 디바이스와 상호 작용할 수 있습니다.

Device Farm의 세션은 웹 브라우저에서 호스팅되는 실제 물리적 디바이스와의 실시간 상호 작용입니다. 세션은 시작할 때 선택한 단일 디바이스를 표시합니다. 사용자는 한 번에 두 개 이상의 세션을 시작할 수 있으며 총 동시 디바이스 수는 사용 중인 디바이스 슬롯 수에 의해 제한됩니다. 디바이스 제품군을 기반으로 디바이스 슬롯을 구입할 수 있습니다(Android 또는 iOS 디바이스). 자세한 내용은 Device Farm 요금을 참조하세요.

Device Farm은 현재 원격 액세스 테스트를 위해 디바이스의 하위 집합을 제공합니다. 새 디바이스는 항상 디바이스 풀에 추가됩니다.

Device Farm은 각 원격 액세스 세션의 비디오를 캡처하고 세션 중에 활동의 로그를 생성합니다. 이 결과에는 세션 중에 제공하는 모든 정보가 포함됩니다.

> **Note**
>
> 보안을 위해 원격 액세스 세션 중에는 계정 번호, 개인 로그인 정보 및 기타 세부 정보 등의 민감한 정보를 제공하거나 입력하지 않는 것이 좋습니다.

주제

- AWS Device Farm에서 원격 액세스 세션 생성
- AWS Device Farm의 원격 액세스 세션 사용
- AWS Device Farm에서 원격 액세스 세션의 결과 가져오기

## AWS Device Farm에서 원격 액세스 세션 생성

원격 액세스 세션에 대한 자세한 내용은 세션 단원을 참조하세요.

- 사전 조건
- 테스트 실행 생성(콘솔)

- 다음 단계

## 사전 조건

- Device Farm에서 프로젝트를 생성하세요. AWS Device Farm에서 프로젝트 생성의 지침을 수행한 다음 이 페이지로 돌아오세요.

## Device Farm 콘솔을 사용하여 세션 만들기

1. https://console.aws.amazon.com/devicefarm에서 Device Farm 콘솔에 로그인하세요.
2. Device Farm 탐색 패널에서 모바일 디바이스 테스트를 선택한 다음 프로젝트를 선택하세요.
3. 프로젝트가 이미 있는 경우 프로젝트 페이지에서 선택하세요. 프로젝트를 만들려면, AWS Device Farm에서 프로젝트 생성의 지침을 따르십시오.
4. 원격 액세스 탭에서 새 세션 시작을 선택하세요.
5. 세션의 디바이스를 선택하세요. 사용 가능한 디바이스 목록에서 선택하거나 목록 맨 위에 있는 필드를 사용하여 디바이스를 검색할 수 있습니다. 검색 기준은 다음과 같습니다.
   - 이름
   - 플랫폼
   - 폼 팩터
   - 플릿 유형
6. 세션 이름에 세션 이름을 입력하세요.
7. 확인 및 세션 시작을 선택하세요.

## 다음 단계

요청한 디바이스를 사용할 수 있게 되면 일반적으로 몇 분 이내에 Device Farm이 세션을 시작합니다. 세션이 시작될 때까지 요청한 디바이스 대화 상자가 나타납니다. 세션 요청을 취소하려면 요청 취소를 선택하세요.

세션이 시작된 후 세션을 중단하지 않고 브라우저 또는 브라우저 탭을 닫아야 하거나 브라우저와 인터넷 간의 연결이 끊어지면 세션은 5분 동안 활성 상태로 유지됩니다. 그런 다음 Device Farm은 세션을 종료합니다. 계정에 유휴 시간에 대한 요금이 청구됩니다.

세션이 시작되면 웹 브라우저에서 디바이스와 상호 작용할 수 있습니다.

# AWS Device Farm의 원격 액세스 세션 사용

원격 액세스 세션을 통해 대화형으로 Android 및 iOS 앱 테스트를 수행하는 방법에 대한 자세한 내용은 세션 단원을 참조하세요.



## 사전 조건

- 세션을 만듭니다. 세션 생성의 지침을 수행한 다음 이 페이지로 돌아오세요.

## Device Farm 콘솔에서 세션 사용

원격 액세스 세션을 요청한 디바이스를 사용할 수 있게 되면 콘솔에 디바이스 화면이 표시됩니다. 세션의 최대 길이는 150분입니다. 세션에 남은 시간은 디바이스 이름 옆의 남은 시간 필드에 표시됩니다.

### 애플리케이션 설치

세션 디바이스에 애플리케이션을 설치하려면 애플리케이션 설치에서 파일 선택을 선택한 후 설치하려는 .apk 파일(Android) 또는 .ipa 파일(iOS)을 선택하세요. 원격 액세스 세션에서 실행되는 애플리케이션에는 테스트 계측이나 프로비저닝이 필요하지 않습니다.

> **Note**
>
> AWS Device Farm은 앱이 설치된 후 확인을 표시하지 않습니다. 앱 아이콘을 통해 앱을 사용할 준비가 되었는지 확인합니다.
> 앱을 업로드해도 사용 가능해지기 전에 시간이 지연되는 경우가 있습니다. 앱을 사용할 수 있는지 보려면 시스템 트레이를 확인하세요.

### 디바이스 제어

터치 마우스 또는 유사한 터치 디바이스 및 디바이스의 화상 키보드를 사용하여 실제 물리적 디바이스처럼 콘솔에 표시된 디바이스와 상호 작용할 수 있습니다. Android 디바이스의 경우 보기 컨트롤에는

Android 디바이스의 홈 및 뒤로 버튼과 같은 버튼이 있습니다. iOS 디바이스의 경우, iOS 디바이스의 홈 버튼과 동일한 기능을 하는 홈 버튼이 있습니다. 최근 앱을 선택하여 디바이스에서 실행 중인 애플리케이션 사이에서 전환할 수도 있습니다.

### 세로 모드와 가로 모드 간 전환

사용 중인 디바이스의 세로(수직) 모드와 가로(수평) 모드 간 전환할 수도 있습니다.

## 다음 단계

Device Farm은 수동으로 중지하거나 150분 제한 시간에 도달할 때까지 세션을 계속 실행합니다. 세션을 종료하려면 세션 중지를 선택하세요. 세션이 중지된 후 캡처된 비디오와 생성된 로그에 액세스할 수 있습니다. 자세한 내용은 세션 결과 가져오기 단원을 참조하세요.

## 팁 및 요령

일부 AWS 리전에서는 원격 액세스 세션에 성능 문제가 발생할 수 있습니다. 이는 부분적으로 일부 리전의 지연 시간 때문입니다. 성능 문제가 발생하면 원격 액세스 세션이 성능을 복구할 수 있는 시간이 경과한 후 앱과 다시 상호 작용합니다.

# AWS Device Farm에서 원격 액세스 세션의 결과 가져오기

세션에 대한 자세한 내용은 세션 단원을 참조하세요.

- 사전 조건
- 세션 세부 정보 보기
- 세션 비디오 또는 로그 다운로드

## 사전 조건

- 세션을 완료하세요. AWS Device Farm의 원격 액세스 세션 사용의 지침을 수행한 다음 이 페이지로 돌아옵니다.

## 세션 세부 정보 보기

원격 액세스 세션이 종료되면 Device Farm 콘솔에 세션 중의 활동에 대한 세부 정보가 포함된 테이블이 표시됩니다. 자세한 내용은 로그 정보 분석을 참조하세요.

나중에 세션의 세부 정보로 돌아가려면 다음을 참조하세요.

1. Device Farm 탐색 패널에서 모바일 장치 테스트를 선택한 다음 프로젝트를 선택하세요.
2. 세션이 포함된 프로젝트를 선택하세요.
3. 원격 액세스를 선택한 다음 목록에서 검토하려는 세션을 선택하세요.

## 세션 비디오 또는 로그 다운로드

원격 액세스 세션이 종료되면 Device Farm 콘솔에서 세션 및 활동 로그의 비디오 캡처에 액세스할 수 있습니다. 세션 결과에서 파일 탭을 선택하면 세션 비디오 및 로그에 대한 링크 목록을 볼 수 있습니다. 브라우저에서 이러한 파일을 보거나 로컬에 저장할 수 있습니다.

# AWS Device Farm에서 프라이빗 디바이스로 작업

프라이빗 디바이스란 AWS Device Farm이 사용자를 대신하여 Amazon 데이터 센터에서 배포하는 물리적 모바일 디바이스입니다. 이 기기는 사용자 AWS 계정에서만 사용할 수 있습니다.

> **Note**
>
> 현재 사설 장치는 AWS 미국 서부 (오레곤) 지역 (`us-west-2`) 에서만 사용할 수 있습니다.

프라이빗 디바이스 플릿이 있으면 프라이빗 디바이스를 사용하여 원격 액세스 세션을 생성하고 테스트 실행을 예약할 수 있습니다. 인스턴스 프로파일을 생성하여 원격 액세스 세션이나 테스트 실행 중에 프라이빗 디바이스의 동작을 제어할 수도 있습니다. 자세한 내용은 AWS Device Farm의 프라이빗 디바이스 관리 단원을 참조하세요. 선택적으로, 특정 Android 프라이빗 디바이스를 루팅된 디바이스로 배포하도록 요청할 수 있습니다.

뿐만 아니라 Amazon Virtual Private Cloud 엔드포인트 서비스를 생성하여 회사가 액세스할 수 있지만 인터넷을 통해 연결할 수 없는 프라이빗 앱을 테스트할 수 있습니다. 예를 들어 모바일 디바이스에서 테스트할 VPC에서 실행 중인 웹 애플리케이션이 있을 수 있습니다. 자세한 내용은 Device Farm - 레거시와 함께 Amazon VPC 엔드포인트 서비스 사용 (권장되지 않음) 단원을 참조하세요.

프라이빗 디바이스가 하나 이상 포함된 플릿을 사용하는 데 관심이 있는 경우 문의하기하세요. Device Farm 팀은 고객과 협력하여 AWS 계정에 사용할 여러 개의 개인 장치를 설정하고 배포해야 합니다.

주제

- AWS Device Farm의 프라이빗 디바이스 관리
- 디바이스 풀에서 프라이빗 디바이스 선택
- AWS Device Farm의 프라이빗 디바이스에서 앱 재서명 건너뛰기
- AWS 리전 간 Amazon VPC 작업
- 사설 기기 종료

## AWS Device Farm의 프라이빗 디바이스 관리

프라이빗 디바이스란 AWS Device Farm이 사용자를 대신하여 Amazon 데이터 센터에서 배포하는 물리적 모바일 디바이스입니다. 이 디바이스는 AWS 계정 전용입니다.

> **Note**
>
> 프라이빗 디바이스는 현재 AWS 미국 서부(오레곤) 리전(us-west-2)에서만 사용할 수 있습니다.

프라이빗 디바이스가 하나 이상 포함된 플릿을 설정할 수 있습니다. 이러한 디바이스는 AWS 계정 전용입니다. 디바이스를 설정한 후 필요에 따라 디바이스에 대한 인스턴스 프로파일을 하나 이상 생성할 수 있습니다. 인스턴스 프로파일을 사용하면 테스트 실행을 자동화하고 디바이스 인스턴스에 동일한 설정을 일관되게 적용할 수 있습니다.

이 주제에서는 인스턴스 프로파일을 생성하고 기타 일반적인 디바이스 관리 작업을 수행하는 방법에 대해 설명합니다.

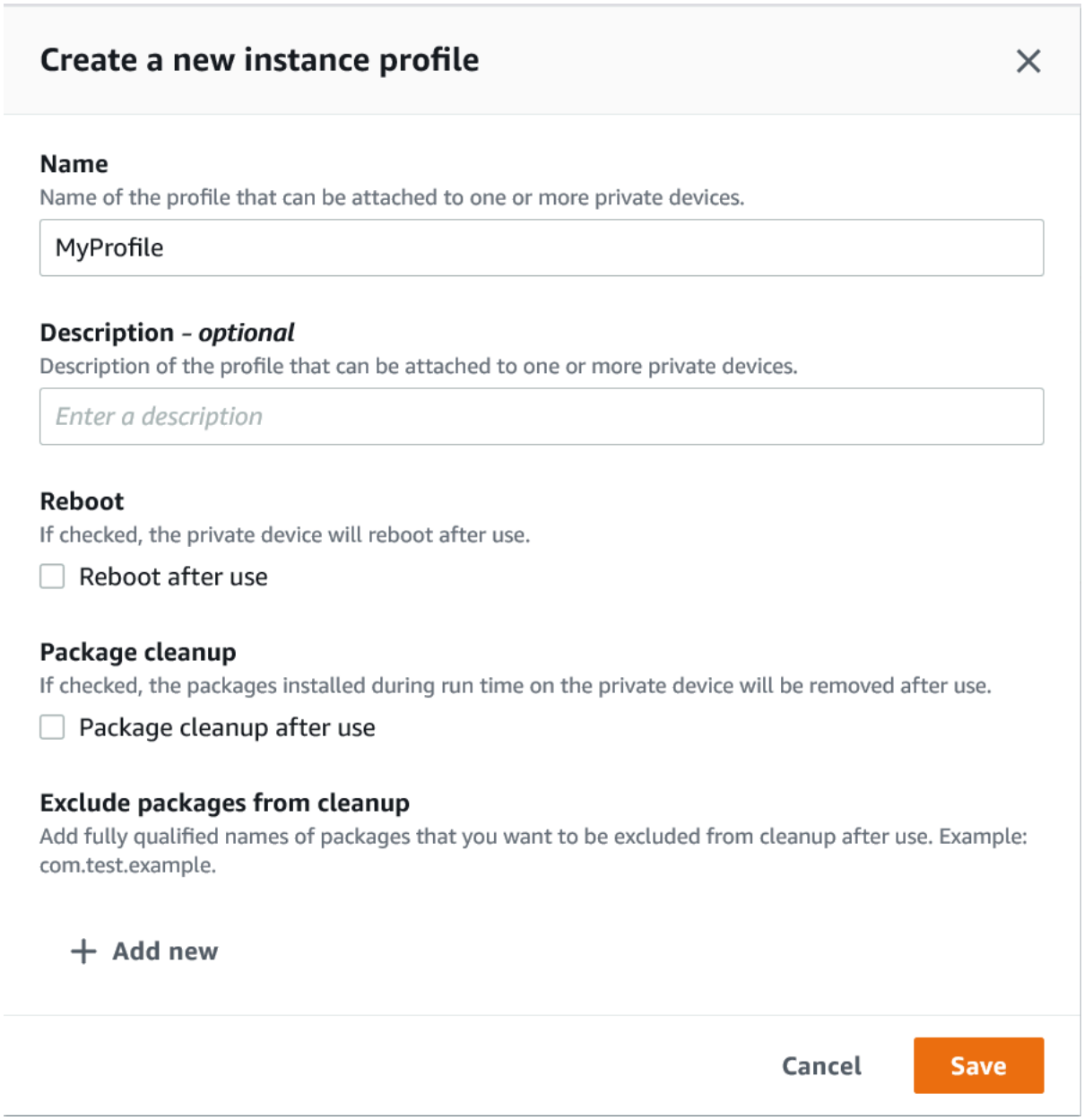
주제



## 인스턴스 프로파일 생성

Device Farm에서 인스턴스 프로파일을 생성하거나 수정하면 테스트 실행이나 원격 액세스 세션 중에 프라이빗 디바이스의 동작을 제어할 수 있습니다. 프라이빗 디바이스 사용을 시작할 때에는 인스턴스 프로파일이 필요하지 않습니다.

1. https://console.aws.amazon.com/devicefarm/에서 Device Farm 콘솔을 여세요.
2. Device Farm 탐색 패널에서 모바일 디바이스 테스트를 선택한 다음 개인 디바이스를 선택하세요.
3. 인스턴스 프로파일을 선택하세요.
4. 새 인스턴스 프로파일 생성을 선택하세요.
5. 인스턴스 프로파일의 이름을 입력하세요.




6. (선택 사항) 인스턴스 프로파일에 대한 설명을 입력하세요.
7. (선택 사항) 다음 설정 중 하나를 변경하여 각 테스트 실행 또는 세션이 종료된 후 이 디바이스에서 수행할 작업을 지정합니다.

   - 사용 후 재부팅: 디바이스를 재부팅하려면 이 확인란을 선택하세요. 기본적으로 이 확인란은 선택 취소되어 있습니다(`false`).
   - 패키지 정리: 디바이스에 설치한 앱 패키지를 모두 제거하려면 이 확인란을 선택하세요. 기본적으로 이 확인란은 선택 취소되어 있습니다(`false`). 디바이스에 설치한 앱 패키지를 모두 유지하려면 이 확인란을 선택 취소된 상태로 두세요.

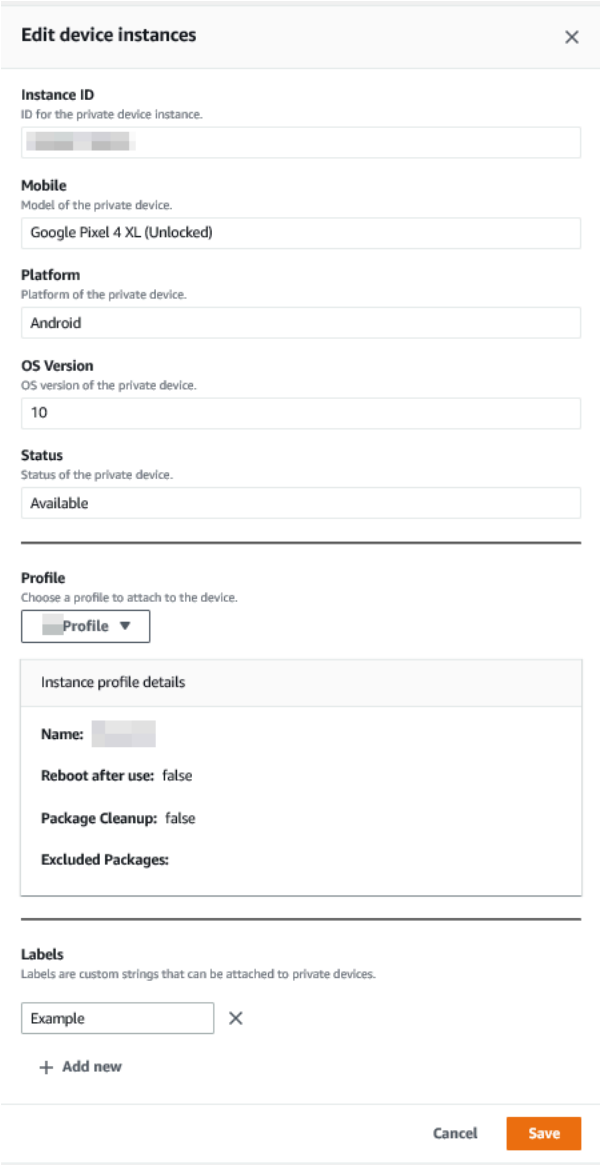
- 정리에서 패키지 제외: 디바이스에서 선택한 앱 패키지만 유지하려면 패키지 정리 확인란을 선택한 다음 새로 추가를 선택하세요. 패키지 이름의 경우 디바이스에 그대로 두려는 앱 패키지의 정규화된 이름을 입력하세요(예: `com.test.example`). 디바이스에 더 많은 앱 패키지를 그대로 두려면 새로 추가를 선택한 다음 각 패키지의 정규화된 이름을 입력하세요.

8. 저장을 선택하세요.

## 프라이빗 디바이스 인스턴스 관리

플릿에 이미 하나 이상의 프라이빗 디바이스가 있는 경우 각 디바이스 인스턴스에 대한 정보를 보고 이에 대한 특정 설정을 관리할 수 있습니다. 추가 프라이빗 디바이스 인스턴스를 요청할 수도 있습니다.

1. https://console.aws.amazon.com/devicefarm/에서 Device Farm 콘솔을 여세요.
2. Device Farm 탐색 패널에서 모바일 디바이스 테스트를 선택한 다음 개인 디바이스를 선택하세요.
3. 디바이스 인스턴스를 선택하세요. 디바이스 인스턴스 탭에는 플릿에 있는 프라이빗 디바이스의 테이블이 표시됩니다. 테이블을 신속하게 검색하거나 필터링하려면 열 위의 필드에 검색어를 입력하십시오.
4. (선택 사항) 새 프라이빗 디바이스 인스턴스를 요청하려면 새 디바이스 인스턴스 요청 또는 문의하기하세요. 프라이빗 디바이스의 경우 Device Farm 팀의 도움으로 추가 설정이 필요합니다.
5. 디바이스 인스턴스의 테이블에서 정보를 보거나 관리하려는 인스턴스 옆의 토글 옵션을 선택하여 수정을 선택하세요.



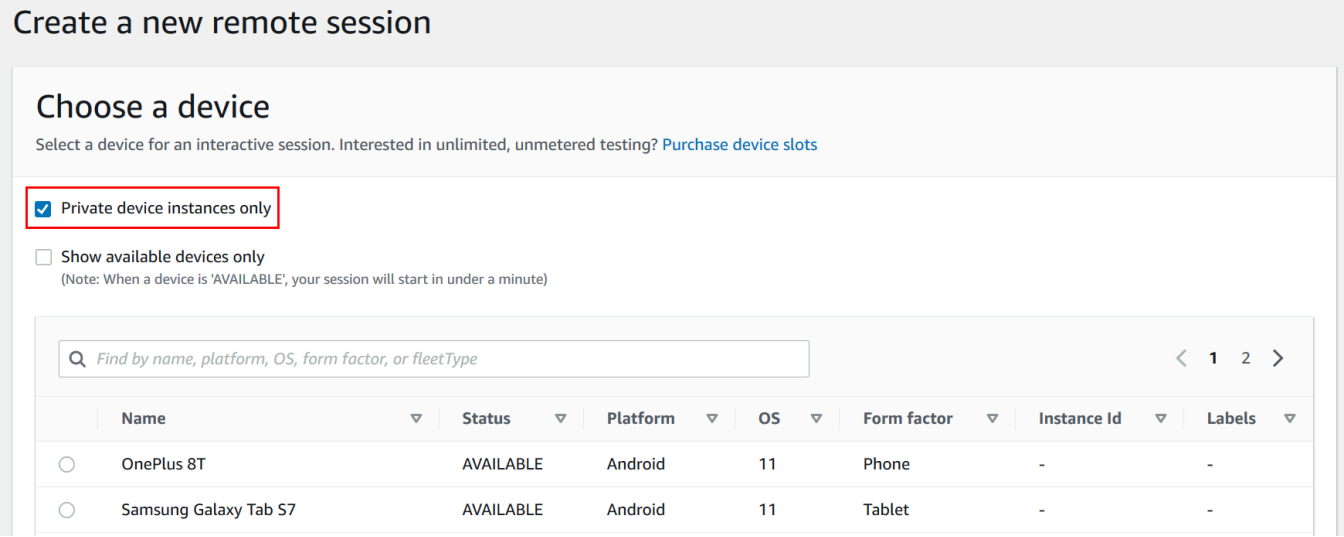

6. (선택 사항) 프로파일에서 디바이스 인스턴스에 연결할 인스턴스 프로파일을 선택하세요. 이는 정리 작업에서 항상 특정 앱 패키지를 제외하려는 경우에 유용할 수 있습니다.
7. (선택 사항) 레이블에서 새로 추가를 선택하여 레이블을 디바이스 인스턴스에 추가합니다. 레이블을 사용하면 디바이스를 분류하고 특정 디바이스를 더 쉽게 찾을 수 있습니다.
8. 저장을 선택하세요.

## 테스트 실행 생성 또는 원격 액세스 세션 시작

프라이빗 디바이스 플릿을 설정한 후 플릿에 있는 프라이빗 디바이스를 하나 이상 사용하여 테스트 실행을 생성하거나 원격 액세스 세션을 시작할 수 있습니다.

1. https://console.aws.amazon.com/devicefarm/에서 Device Farm 콘솔을 여세요.
2. Device Farm 탐색 패널에서 모바일 디바이스 테스트를 선택한 다음 프로젝트를 선택하세요.
3. 목록에서 기존 프로젝트를 선택하거나 새 프로젝트를 생성합니다. 새 프로젝트를 생성하려는 경우 새 프로젝트를 선택하고 프로젝트 이름을 입력한 다음 제출을 선택하세요.
4. 다음 중 하나를 수행하세요.

   - 테스트 실행을 생성하려면 자동 테스트를 선택한 다음 새 실행 생성을 선택하세요. 이 마법사에서는 실행 생성 단계를 안내합니다. 디바이스 선택 단계에서는 기존 디바이스 풀을 편집하거나 Device Farm 팀이 설정하여 AWS 계정과 연결한 해당 프라이빗 디바이스만 포함하는 새 디바이스 풀을 생성할 수 있습니다. 자세한 내용은 the section called “프라이빗 디바이스 풀 생성” 단원을 참조하세요.
   - 원격 액세스 세션을 시작하려면 원격 액세스를 선택한 다음 새 세션 시작을 선택하세요. 디바이스 선택 페이지에서 프라이빗 디바이스 인스턴스만을 선택하여 Device Farm 팀이 설정하여 AWS 계정과 연결한 해당 프라이빗 디바이스로만 목록을 제한합니다. 그런 다음 액세스할 디바이스를 선택하고 원격 액세스 세션의 이름을 입력한 다음 확인 및 세션 시작을 선택하세요.




## 다음 단계

프라이빗 디바이스를 설정한 후에는 다음과 같은 방법으로 프라이빗 디바이스를 관리할 수도 있습니다.

- 프라이빗 디바이스에서 앱 다시 서명 건너뛰기
- Amazon Virtual Private Cloud 엔드포인트 서비스를 Device Farm과 함께 사용

인스턴스 프로파일을 삭제하려면 인스턴트 프로파일 메뉴에서 삭제하려는 인스턴스 옆의 토글 옵션을 선택한 다음 삭제를 선택하세요.

# 디바이스 풀에서 프라이빗 디바이스 선택

테스트 실행에서 프라이빗 디바이스를 사용하기 위해 프라이빗 디바이스를 선택하는 디바이스 풀을 생성할 수 있습니다. 디바이스 풀을 사용하면 주로 세 가지 유형의 디바이스 풀 규칙을 통해 프라이빗 디바이스를 선택할 수 있습니다.

1. 디바이스 ARN 기반 규칙
2. 디바이스 인스턴스 레이블 기반 규칙
3. 디바이스 인스턴스 ARN 기반 규칙

다음 섹션에서는 각 규칙 유형과 사용 사례를 자세히 설명합니다. Device Farm 콘솔, AWS 명령줄 인터페이스(AWS CLI) 또는 Device Farm API를 사용하여 이러한 규칙을 사용하는 프라이빗 디바이스를 포함하는 디바이스 풀을 생성하거나 수정할 수 있습니다.

주제

- 디바이스 ARN
- 디바이스 인스턴스 레이블
- 인스턴스 ARN
- 프라이빗 디바이스가 포함된 프라이빗 디바이스 풀 생성(콘솔)
- 프라이빗 디바이스를 사용하여 프라이빗 디바이스 풀 생성(AWS CLI)
- 프라이빗 디바이스(API) 를 사용하여 프라이빗 디바이스 풀 생성

## 디바이스 ARN

디바이스 ARN은 특정 물리적 디바이스 인스턴스가 아닌 디바이스 유형을 나타내는 식별자입니다. 디바이스 유형은 다음 속성으로 정의됩니다.

- 디바이스의 플릿 ID
- 디바이스의 OEM
- 디바이스의 모델 번호
- 디바이스의 운영 체제 버전

- 루팅 여부를 나타내는 디바이스 상태

많은 물리적 디바이스 인스턴스는 단일 디바이스 유형으로 표시될 수 있으며, 이때 해당 유형의 모든 인스턴스는 이러한 속성에 대해 동일한 값을 갖습니다. 예를 들어 개인 플릿에 iOS 버전 *16.1.0*의 *Apple iPhone 13* 디바이스 3대가 있는 경우 각 디바이스는 동일한 디바이스 ARN을 공유하게 됩니다. 동일한 속성을 가진 디바이스가 플릿에서 추가 또는 제거된 경우 디바이스 ARN은 플릿에서 해당 디바이스 유형에 사용할 수 있는 모든 디바이스를 계속 나타냅니다.

디바이스 ARN은 어떤 시점에 배포한 특정 디바이스 인스턴스와 상관없이 디바이스 풀에서 디바이스를 계속 선택할 수 있도록 하기 때문에 디바이스 풀에 사용할 프라이빗 디바이스를 선택하는 가장 강력한 방법입니다. 개별 프라이빗 디바이스 인스턴스에서 하드웨어 장애가 발생할 수 있으며, 이 경우 Device Farm은 해당 인스턴스를 동일한 디바이스 유형의 새 작동 인스턴스로 자동으로 교체합니다. 이러한 시나리오에서 디바이스 ARN 규칙은 하드웨어 장애 발생 시 디바이스 풀에서 디바이스를 계속 선택할 수 있도록 합니다.

디바이스 풀의 프라이빗 디바이스에 디바이스 ARN 규칙을 사용하고 해당 풀로 테스트 실행을 예약하면 Device Farm이 해당 디바이스 ARN이 나타내는 프라이빗 디바이스 인스턴스를 자동으로 확인합니다. 현재 사용 가능한 인스턴스 중 하나가 테스트를 실행하도록 할당됩니다. 현재 사용 가능한 인스턴스가 없는 경우 Device Farm은 해당 디바이스 ARN의 사용 가능한 첫 인스턴스가 제공될 때까지 기다렸다가 테스트를 실행하도록 할당합니다.

## 디바이스 인스턴스 레이블

디바이스 인스턴스 레이블은 디바이스 인스턴스의 메타데이터로 첨부할 수 있는 텍스트 식별자입니다. 각 디바이스 인스턴스에 여러 레이블을 부착하고 여러 디바이스 인스턴스에 동일한 레이블을 부착할 수 있습니다. 디바이스 인스턴스에서 디바이스 레이블을 추가, 수정 또는 제거하는 방법에 대한 자세한 내용은 프라이빗 디바이스 관리를 참조하세요.

디바이스 인스턴스 레이블은 디바이스 풀의 프라이빗 디바이스를 선택하는 강력한 방법이 될 수 있습니다. 레이블이 같은 디바이스 인스턴스가 여러 개 있는 경우 디바이스 풀이 이 중 하나를 선택하여 테스트할 수 있기 때문입니다. 디바이스 ARN이 사용 사례에 적합하지 않은 경우(예: 여러 디바이스 유형의 디바이스 중에서 선택하거나 디바이스 유형의 모든 디바이스 하위 집합에서 선택하려는 경우) 디바이스 인스턴스 레이블을 사용하면 디바이스 풀의 여러 디바이스 중에서 더 세부적으로 선택할 수 있습니다. 개별 프라이빗 디바이스 인스턴스에서 하드웨어 장애가 발생할 수 있으며, 이 경우 Device Farm은 해당 인스턴스를 동일한 디바이스 유형의 새 작동 인스턴스로 자동으로 교체합니다. 이러한 시나리오에서 교체 디바이스 인스턴스는 교체된 디바이스의 인스턴스 레이블 메타데이터를 보존하지 않습니다. 따라서 여러 디바이스 인스턴스에 동일한 디바이스 인스턴스 레이블을 적용하는 경우 디바이스 인

스턴스 레이블 규칙은 하드웨어 장애 발생 시 디바이스 풀에서 디바이스 인스턴스를 계속 선택할 수 있도록 합니다.

디바이스 풀의 프라이빗 디바이스에 디바이스 인스턴스 레이블 규칙을 사용하고 해당 풀로 테스트 실행을 예약하면 Device Farm은 자동으로 어떤 프라이빗 디바이스 인스턴스가 해당 디바이스 인스턴스 레이블로 표시되는지 확인하고, 해당 인스턴스 중에서 테스트를 실행하는 데 사용할 수 있는 인스턴스를 임의로 선택합니다. 사용할 수 없는 경우 Device Farm은 테스트를 실행할 디바이스 인스턴스 레이블이 있는 디바이스 인스턴스를 임의로 선택하고, 사용 가능한 상태가 되면 디바이스에서 실행되도록 대기열에 넣습니다.

## 인스턴스 ARN

디바이스 인스턴스 ARN은 프라이빗 플릿에 배포된 물리적 베어 메탈 디바이스 인스턴스를 나타내는 식별자입니다. 예를 들어 프라이빗 플릿에 OS *15.0.0*가 설치된 *iPhone 13* 디바이스 3대가 동일한 디바이스 ARN을 공유하는 경우, 각 디바이스는 해당 인스턴스만을 나타내는 자체 인스턴스 ARN도 갖게 됩니다.

디바이스 인스턴스 ARN은 디바이스 풀에 사용할 프라이빗 디바이스를 선택하는 가장 강력한 방법이 아니므로 디바이스 ARN과 디바이스 인스턴스 레이블이 사용 사례에 맞지 않는 경우에만 사용하는 것이 좋습니다. 테스트의 전제 조건으로 특정 디바이스 인스턴스가 고유하고 특정한 방식으로 구성되어 있고 테스트를 실행하기 전에 해당 구성을 알고 확인해야 하는 경우 디바이스 인스턴스 ARN이 디바이스 풀의 규칙으로 사용되는 경우가 많습니다. 개별 프라이빗 디바이스 인스턴스에서 하드웨어 장애가 발생할 수 있으며, 이 경우 Device Farm은 해당 인스턴스를 동일한 디바이스 유형의 새 작동 인스턴스로 자동으로 교체합니다. 이러한 시나리오에서 교체 디바이스 인스턴스의 디바이스 인스턴스 ARN은 교체된 디바이스와 다릅니다. 따라서 디바이스 풀에 디바이스 인스턴스 ARN을 사용하는 경우 디바이스 풀의 규칙 정의를 기존 ARN 사용에서 새 ARN 사용으로 수동으로 변경해야 합니다. 테스트를 위해 디바이스를 수동으로 사전 구성해야 하는 경우 디바이스 ARN에 비해 효과적인 워크플로가 될 수 있습니다. 대규모 테스트를 위해서는 디바이스 인스턴스 레이블에 맞게 이러한 사용 사례를 조정하고, 가능하면 테스트용으로 여러 디바이스 인스턴스를 미리 구성하는 것이 좋습니다.

디바이스 풀의 프라이빗 디바이스에 디바이스 인스턴스 ARN 규칙을 사용하고 해당 풀로 테스트 실행을 예약하면 Device Farm에서 해당 테스트를 해당 디바이스 인스턴스에 자동으로 할당합니다. 해당 디바이스 인스턴스를 사용할 수 없는 경우 Device Farm은 사용 가능한 상태가 되면 해당 디바이스에서 테스트를 대기열에 넣습니다.

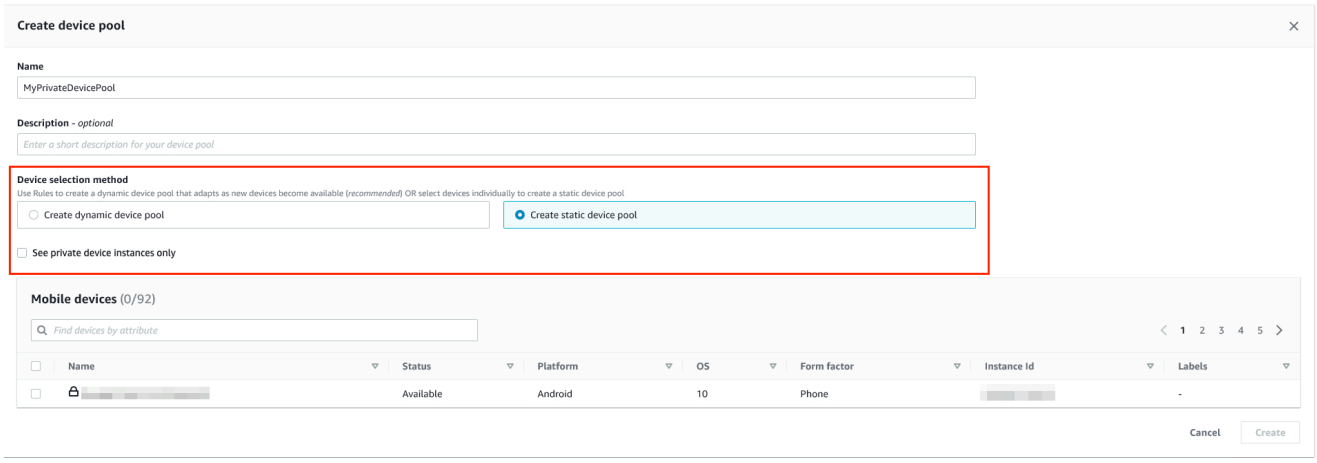
## 프라이빗 디바이스가 포함된 프라이빗 디바이스 풀 생성(콘솔)

테스트 실행을 생성하면 테스트 실행을 위한 디바이스 풀을 생성하고 풀에 프라이빗 디바이스만 포함되어 있는지 확인할 수 있습니다.

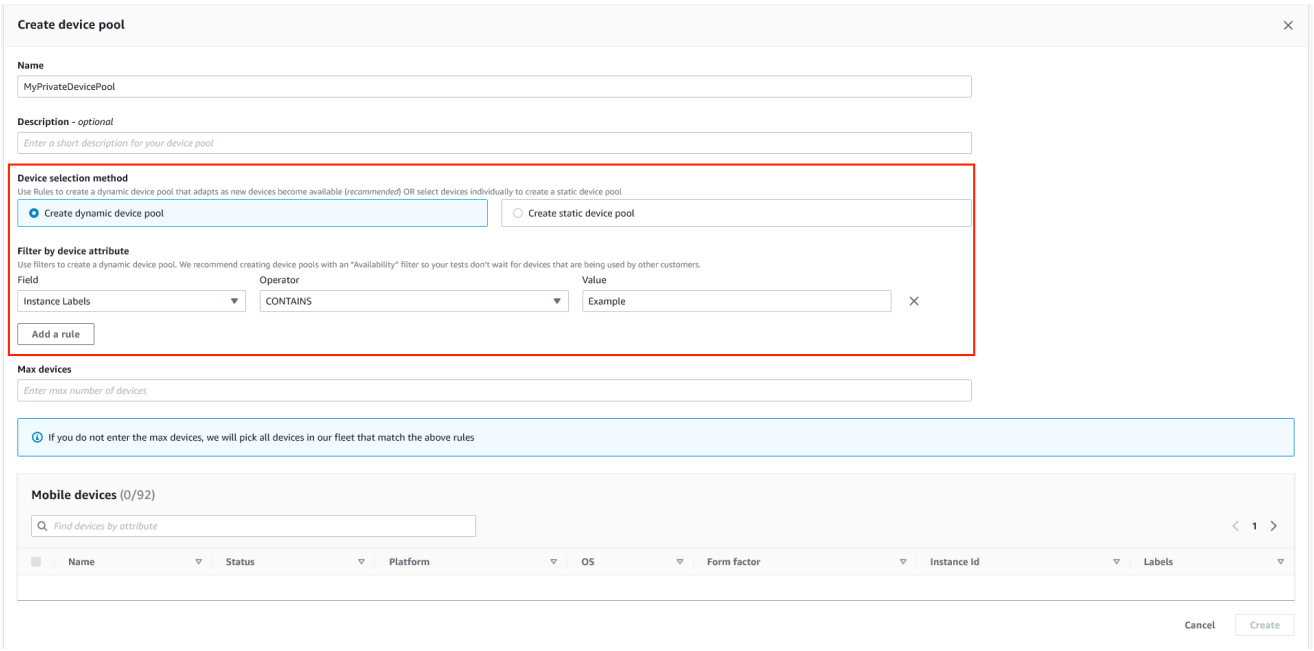
> **Note**
>
> 콘솔에서 프라이빗 디바이스가 포함된 디바이스 풀을 생성할 때는 사용 가능한 세 가지 규칙 중 하나만 사용하여 프라이빗 디바이스를 선택할 수 있습니다. 프라이빗 디바이스에 대한 여러 유형의 규칙이 포함된 디바이스 풀을 생성하려면(예: 디바이스 ARN 및 디바이스 인스턴스 ARN에 대한 규칙이 포함된 디바이스 풀) CLI 또는 API를 통해 풀을 생성해야 합니다.

1. https://console.aws.amazon.com/devicefarm/에서 Device Farm 콘솔을 엽니다.
2. Device Farm 탐색 패널에서 모바일 디바이스 테스트를 선택한 다음 프로젝트를 선택하세요.
3. 목록에서 기존의 프로젝트를 선택하거나 새로운 프로젝트를 생성하세요. 새 프로젝트를 생성하는 경우 새 프로젝트를 선택한 후 프로젝트의 이름을 입력하고 제출을 선택하세요.
4. 자동 테스트를 선택한 다음 새 실행 생성을 선택하세요. 이 마법사에서는 애플리케이션을 선택하고 실행할 테스트를 구성하는 단계를 안내합니다.
5. 디바이스 선택 단계에서 새 디바이스 풀 생성을 선택하고 디바이스 풀의 이름 및 설명(선택 사항)을 입력하세요.

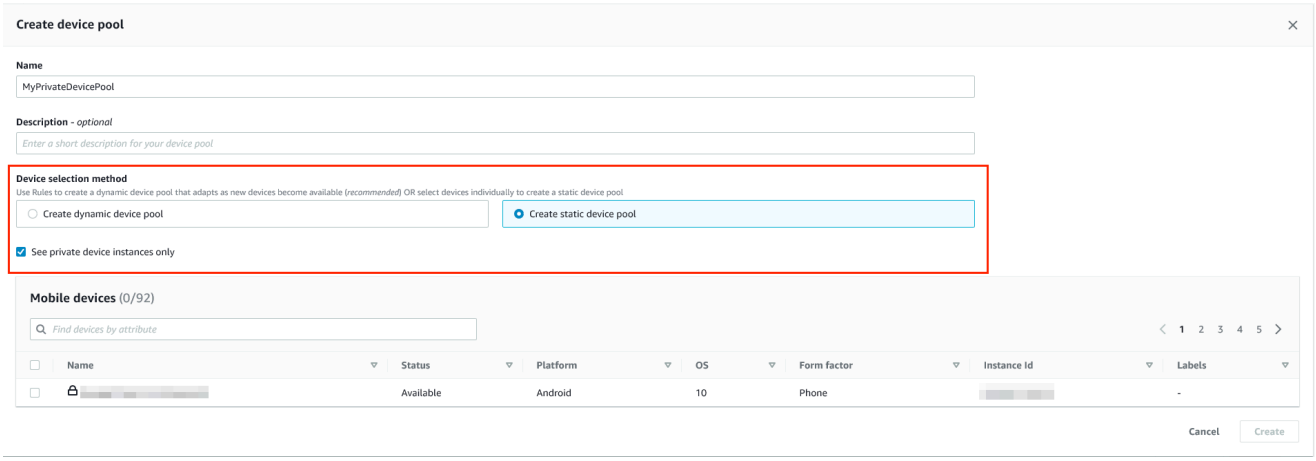
   a. 디바이스 풀에 디바이스 ARN 규칙을 사용하려면 정적 디바이스 풀 생성을 선택한 다음 목록에서 디바이스 풀에서 사용할 특정 디바이스 유형을 선택하세요. 이 옵션을 선택하면 디바이스 ARN 규칙 대신 디바이스 인스턴스 ARN 규칙을 사용하여 디바이스 풀이 생성되므로 프라이빗 디바이스 인스턴스만을 선택하지 마세요.




   b. 디바이스 풀에 디바이스 인스턴스 레이블 규칙을 사용하려면 동적 디바이스 풀 생성을 선택하세요. 그런 다음 디바이스 풀에서 사용하려는 각 레이블에 대해 규칙 추가를 선택하세요. 각 규칙에 대해 인스턴스 레이블을 `Field`로 선택하고 포함을 `Operator`로 선택한 다음 원하는 디바이스 인스턴스 레이블을 `Value`로 지정하세요.




c. 디바이스 풀에 디바이스 인스턴스 ARN 규칙을 사용하려면 정적 디바이스 풀 생성을 선택한 다음 프라이빗 디바이스 인스턴스만을 선택하여 디바이스 목록을 Device Farm이 사용자 AWS 계정에 연결한 프라이빗 디바이스 인스턴스로만 제한하세요.




6. 생성을 선택하세요.

## 프라이빗 디바이스를 사용하여 프라이빗 디바이스 풀 생성(AWS CLI)

- create-device-pool 명령을 실행하세요.

AWS CLI의 Device Farm 사용 방법에 대한 자세한 내용은 AWS CLI 참조 단원을 참조하세요.

## 프라이빗 디바이스(API) 를 사용하여 프라이빗 디바이스 풀 생성

- `CreateDevicePool` API를 호출하세요.

Device Farm API 사용에 대한 자세한 내용은 Device Farm 자동화 단원을 참조하세요.

# AWS Device Farm의 프라이빗 디바이스에서 앱 재서명 건너뛰기

앱 서명은 기기에 설치하거나 Google Play 스토어 또는 Apple App Store와 같은 앱 스토어에 게시하기 전에 개인 키를 사용하여 앱 패키지 (예: APK, IPA) 에 디지털 서명하는 프로세스입니다. 필요한 서명과 프로필 수를 줄여 테스트를 간소화하고 원격 디바이스의 데이터 보안을 강화하기 위해 AWS Device Farm은 앱이 서비스에 업로드된 후 다시 서명합니다.

앱을 AWS Device Farm에 업로드하면 서비스가 자체 서명 인증서 및 프로비저닝 프로필을 사용하여 앱에 대한 새 서명을 생성합니다. 이 프로세스는 원래 앱 서명을 AWS Device Farm의 서명으로 대체합니다. 그러면 다시 서명된 앱이 AWS Device Farm에서 제공하는 테스트 디바이스에 설치됩니다. 새 서명을 사용하면 원래 개발자 인증서 없이도 이러한 디바이스에서 앱을 설치하고 실행할 수 있습니다.

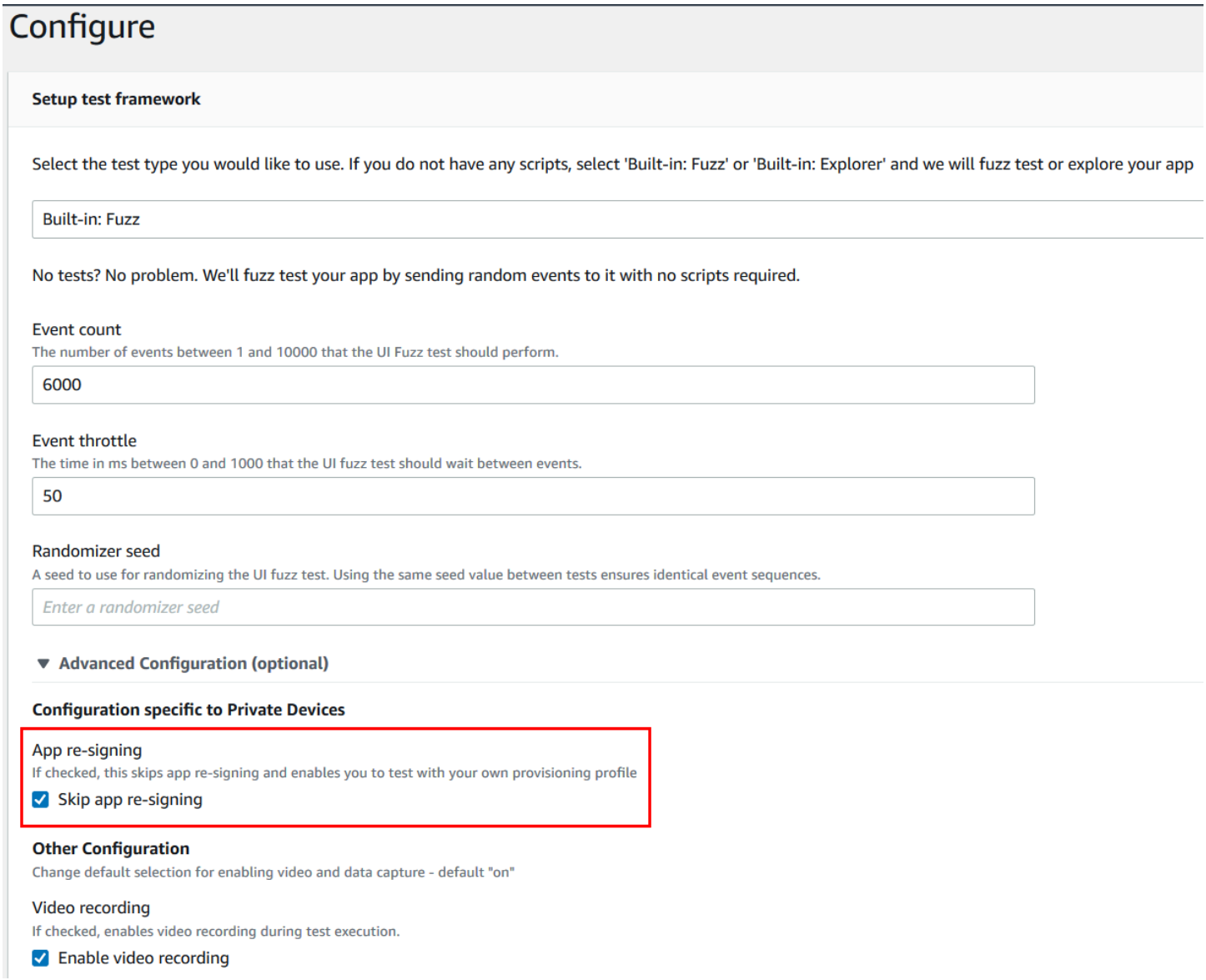
iOS에서는 내장된 프로비저닝 프로필을 와일드카드 프로필로 바꾸고 앱을 다시 디자인합니다. 데이터를 제공하면 설치 전에 애플리케이션 패키지에 보조 데이터를 추가하여 해당 데이터가 앱의 샌드박스에 표시되도록 합니다. iOS 앱을 재서명하면 특정 권한이 제거됩니다. 여기에는 앱 그룹, 관련 도메인, 게임 센터,, 무선 액세서리 구성 HealthKit HomeKit, 인앱 구매, 앱 간 오디오, Apple Pay, 푸시 알림, VPN 구성 및 제어가 포함됩니다.

Android에서는 앱을 다시 종료합니다. 이로 인해 앱 서명에 의존하는 기능 (예: Google Maps Android API) 이 작동하지 않을 수 있습니다. 또한 다음과 같은 제품에서 사용할 수 있는 불법 복제 방지 및 변조 방지 탐지가 트리거될 수 있습니다. DexGuard 기본 제공 테스트의 경우 스크린샷을 캡처하고 저장하는 데 필요한 권한을 포함하도록 매니페스트를 수정할 수 있습니다.

프라이빗 디바이스를 사용하는 경우 AWS Device Farm이 앱을 재서명하는 단계를 건너뛸 수 있습니다. 이는 Device Farm이 항상 Android 및 iOS 플랫폼에서 앱에 재서명하는 공용 디바이스와는 다릅니다.

원격 액세스 세션 또는 테스트 실행을 만들 때 앱 재서명을 건너뛸 수 있습니다. 이는 Device Farm이 앱을 재서명할 때 앱 기능이 중단되는 경우 유용할 수 있습니다. 예를 들어, 재서명 후에는 푸시 알림이 작동하지 않을 수 있습니다. Device Farm이 앱을 테스트할 때 변경하는 사항에 대한 자세한 내용은 AWS Device Farm FAQ 또는 앱 페이지를 참조하십시오.

테스트 실행을 위한 앱 재서명을 건너뛰려면 테스트 실행을 생성할 때 구성 페이지에서 앱 재서명 건너뛰기를 선택하세요.




> **Note**
>
> XCTest 프레임워크를 사용하는 경우 앱 재서명 건너뛰기 옵션을 사용할 수 없습니다. 자세한 내용은 iOS용 XCTest 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

앱 서명 설정을 구성하는 추가 단계는 프라이빗 Android 또는 iOS 디바이스를 사용하는지 여부에 따라 달라집니다.

## Android 디바이스에서 앱 재서명 건너뛰기

프라이빗 Android 디바이스에서 앱을 테스트하는 경우 테스트 실행 또는 원격 액세스 세션을 만들 때 앱 재서명 건너뛰기를 선택하세요. 추가 구성이 필요하지 않습니다.

## iOS 디바이스에서 앱 재서명 건너뛰기

Apple은 테스트를 위해 앱을 디바이스에 로드하기 전에 앱에 서명하도록 요구합니다. iOS 디바이스의 경우 앱 서명을 위한 2가지 옵션이 있습니다.

- 사내(엔터프라이즈) 개발자 프로필을 사용하는 경우 다음 the section called “앱을 신뢰할 수 있는 원격 액세스 세션 생성” 단원으로 건너뛰어도 됩니다.

- 임시 iOS 앱 개발 프로필을 사용하는 경우 먼저 Apple 개발자 계정으로 디바이스를 등록한 다음 프라이빗 디바이스를 포함하도록 프로비저닝 프로필을 업데이트해야 합니다. 그런 다음 업데이트한 프로비저닝 프로필을 사용하여 앱에 재서명해야 합니다. 이후 Device Farm에서 재서명된 앱을 실행할 수 있습니다.

임시 iOS 앱 개발 프로비전 프로필에 디바이스를 등록하려면 다음을 참조하세요.

1. Apple 개발자 계정에 로그인하세요.
2. 콘솔의 인증서, ID, 프로필 섹션으로 이동하세요.
3. 디바이스로 이동하세요.
4. Apple 개발자 계정에 테스트 디바이스를 등록하세요. 디바이스의 이름과 UDID를 가져오려면 Device Farm API의 `ListDeviceInstances` 작업을 사용하세요.
5. 프로비저닝 프로필로 이동하여 편집을 선택하세요.
6. 목록에서 디바이스를 선택하세요.
7. Xcode에서 업데이트된 프로비저닝 프로필을 가져온 다음 앱에 재서명하세요.

추가 구성이 필요하지 않습니다. 이제 원격 액세스 세션 또는 테스트 실행을 생성하고 앱 재서명 건너뛰기를 선택할 수 있습니다.

## iOS 앱을 신뢰할 수 있는 원격 액세스 세션 생성

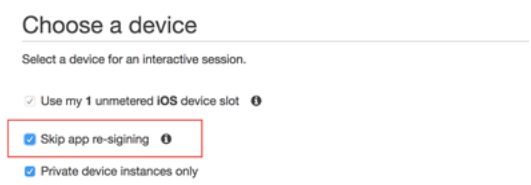
사내(엔터프라이즈) 개발자 프로비저닝 프로필을 사용하는 경우 각 개인 디바이스에서 사내 앱 개발자 인증서를 신뢰하는 일회성 절차를 수행해야 합니다.

이렇게 하려면 테스트하려는 앱을 프라이빗 디바이스에 설치하거나 테스트하려는 앱과 동일한 인증서로 서명된 더미 앱을 설치할 수 있습니다. 동일한 인증서로 서명된 더미 앱을 설치하면 이점이 있습니다. 구성 프로필 또는 엔터프라이즈 앱 개발자를 신뢰하면, 삭제 전까지 해당 개발자의 모든 앱을 프라이빗 디바이스에서 신뢰할 수 있습니다. 따라서 테스트하려는 앱의 새 버전을 업로드할 때 앱 개발자를 다시 신뢰하지 않아도 됩니다. 이는 테스트 자동화를 실행하고 앱을 테스트할 때마다 원격 액세스 세션을 만들지 않으려는 경우에 특히 유용합니다.

원격 액세스 세션을 시작하기 전에 인스턴스 프로파일 생성의 단계에 따라 Device Farm에서 인스턴스 프로파일을 만들거나 수정하세요. 인스턴스 프로파일에서 테스트 앱 또는 더미 앱의 번들 ID를 정리에서 패키지 제외 설정에 추가하세요. 그런 다음 인스턴스 프로파일을 프라이빗 디바이스 인스턴스에 연결하여 새 테스트 실행을 시작하기 전에 Device Farm이 디바이스에서 이 앱을 제거하지 않도록 합니다. 이렇게 하면 개발자 인증서를 계속 신뢰할 수 있습니다.

원격 액세스 세션을 사용하여 더미 앱을 디바이스에 업로드하면 앱을 시작하고 개발자를 신뢰할 수 있습니다.

1. 세션 생성의 지침에 따라 생성한 프라이빗 디바이스 인스턴스 프로파일을 사용하는 원격 액세스 세션을 생성하세요. 세션을 생성할 때는 앱 재서명 건너뛰기를 선택해야 합니다.




> **Important**
>
> 디바이스 목록을 필터링하여 프라이빗 디바이스만 포함하려면 프라이빗 디바이스 인스턴스만을 선택하여 올바른 인스턴스 프로파일이 있는 프라이빗 디바이스를 사용할 수 있도록 하세요.

   또한 테스트하려는 더미 앱 또는 앱을 이 인스턴스에 연결된 인스턴스 프로파일의 정리에서 패키지 제외 설정에 추가해야 합니다.
2. 원격 세션이 시작되면 파일 선택을 선택하여 사내 프로비저닝 프로필을 사용하는 애플리케이션을 설치하세요.
3. 방금 업로드한 앱을 실행하세요.
4. 지침에 따라 개발자 인증서를 신뢰하세요.

이제 이 구성 프로필 또는 엔터프라이즈 앱 개발자의 모든 앱은 삭제 전까지 이 프라이빗 디바이스에서 신뢰됩니다.

# AWS 리전 간 Amazon VPC 작업

Device Farm 서비스는 미국 서부(오레곤)(us-west-2) 리전에만 있습니다. Amazon Virtual Private Cloud(Amazon VPC)를 사용하면 Device Farm을 사용하는 다른 AWS 리전의 Amazon Virtual Private Cloud 내 서비스에 연결할 수 있습니다. Device Farm과 서비스가 같은 리전에 있는 경우 Device Farm - 레거시와 함께 Amazon VPC 엔드포인트 서비스 사용 (권장되지 않음)을 참조하세요.

다른 리전에 있는 개인 서비스에 액세스하는 방법은 두 가지입니다. us-west-2이 아닌 리전에 서비스가 있는 경우 VPC 피어링을 사용하여 해당 리전의 VPC를 us-west-2의 Device Farm과 연결되는 VPC와 피어링할 수 있습니다. 그러나 여러 리전에 서비스가 있는 경우 Transit Gateway를 사용하면 더 간단한 네트워크 구성으로 해당 서비스에 액세스할 수 있습니다.

자세한 내용은 Amazon VPC 피어링 가이드의 VPC 피어링 시나리오를 참조하세요.

## VPC 피어링

중복되지 않는 고유한 CIDR 블록이 있다면 서로 다른 리전의 두 VPC를 피어링할 수 있습니다. 이를 통해 모든 개인 IP 주소가 고유하며, 어떠한 형태의 Network Address Translation(NAT)도 필요 없이 VPC의 모든 리소스가 서로 주소를 지정할 수 있습니다. CIDR 표기법에 대한 자세한 정보는 RFC 4632을 참조하세요.

이 항목에는 Device Farm(VPC-1로 언급됨)이 미국 서부(오레곤)(us-west-2)에 위치한 경우의 크로스 리전 예제 시나리오를 포함합니다. 이 예제의 두 번째 VPC(VPC-2로 일컬어 짐)는 다른 리전 있습니다.

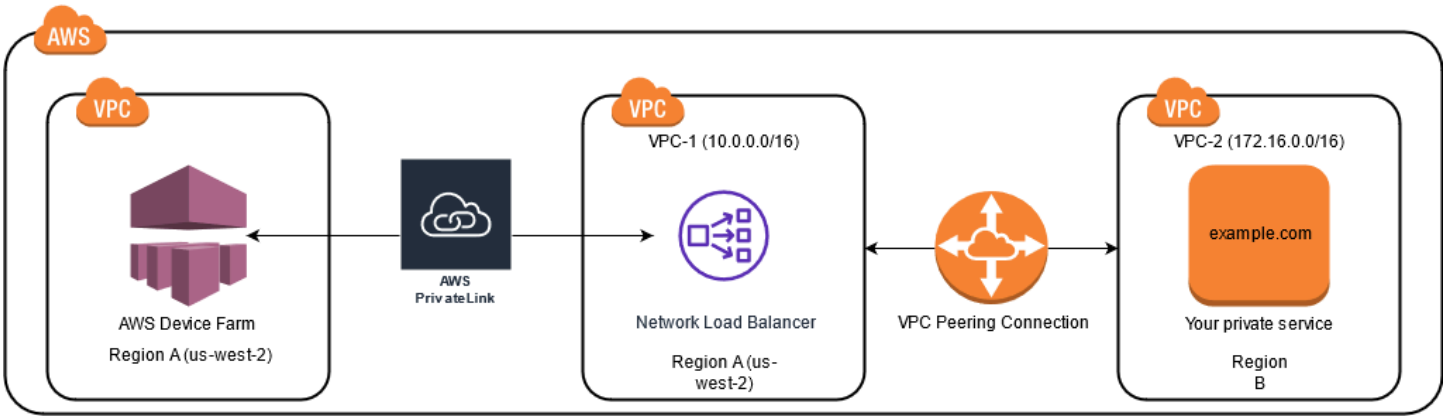
Device Farm VPC 리전 간 예제

| VPC 구성 요소 | VPC-1 | VPC-2 |
| --- | --- | --- |
| CIDR | 10.0.0.0/16 | 172.16.0.0/16 |

> **Important**
>
> 두 VPC 간에 피어링 연결을 설정하면 VPC의 보안 상태가 변경될 수 있습니다. 또한, 라우팅 테이블에 새 항목을 추가하면 VPC 내 리소스의 보안 상태가 변경될 수 있습니다. 보안 요구 사

항을 충족하는 환경 설정 구현은 사용자에 달려 있습니다. 자세한 내용은 공동 책임 모델을 참조하세요.

다음 다이어그램은 예제의 구성 요소와 구성 요소 간의 상호 작용을 보여줍니다.




주제



## 사전 조건

이 예제는 다음을 필요로 합니다.

- 겹치지 않는 CIDR 블록을 포함하는 서브넷으로 구성된 두 개의 VPC
- VPC-1은 `us-west-2` 리전에 있어야 하고 가용 영역 `us-west-2a`, `us-west-2b`, `us-west-2c`에 대한 서브넷을 포함해야 합니다.

VPC 생성 및 서브넷 구성에 대한 자세한 내용은 Amazon VPC 피어링 가이드의 VPC 및 서브넷 사용을 참조하세요.

## 1단계: VPC-1 및 VPC-2 간의 피어링 연결 설정

겹치지 않는 CIDR 블록을 포함하는 두 VPC 간에 피어링 연결을 설정하세요. 이를 위해서는 Amazon VPC 피어링 가이드의 VPC 피어링 연결 생성 및 수락을 참조하세요. 이 주제의 크로스 리전 시나리오와 Amazon VPC 피어링 가이드를 사용하여 다음과 같은 예제 피어링 연결 구성을 생성하세요.

이름

```
Device-Farm-Peering-Connection-1
```

VPC ID(요청자)

```
vpc-0987654321gfedcba (VPC-2)
```

계정

```
My account
```

리전

```
US West (Oregon) (us-west-2)
```

VPC ID(수락자)

```
vpc-1234567890abcdefg (VPC-1)
```

> **Note**
>
> 새 피어링 연결을 설정할 때는 반드시 VPC 피어링 연결 할당을 참조하세요. 자세한 내용은 Amazon VPC 사용 설명서의 Amazon VPC 할당을 참조하세요.

## 2단계: VPC-1 및 VPC-2 내 라우팅 테이블 업데이트

피어링 연결을 설정한 후에는 두 VPC 간에 데이터를 전송할 목적지 경로를 설정해야 합니다. 이 경로를 설정하려면 VPC-1 라우팅 테이블을 수동으로 업데이트하여 VPC-2 서브넷을 가리키거나 그 반대로 진행합니다. 이렇게 하려면 Amazon VPC 피어링 가이드의 VPC 피어링 연결에 대한 라우팅 테이블 업데이트를 참조하십시오. 이 주제의 크로스 리전 시나리오와 Amazon VPC 피어링 가이드를 사용하여 다음과 같은 예제 라우팅 테이블 구성을 생성하세요.

Device Farm VPC 라우팅 테이블 예제

| VPC 구성 요소 | VPC-1 | VPC-2 |
| --- | --- | --- |
| 라우팅 테이블 ID | rtb-1234567890abcdefg | rtb-0987654321gfedcba |
| 로컬 주소 범위 | 10.0.0.0/16 | 172.16.0.0/16 |
| 대상 주소 범위 | 172.16.0.0/16 | 10.0.0.0/16 |

## 3단계: 대상 그룹 생성

대상 경로를 설정한 후 VPC-1 내의 Network Load Balancer를 구성하여 요청을 VPC-2로 라우팅할 수 있습니다.

Network Load Balancer는 먼저 요청이 전송되는 IP 주소를 포함하는 대상 그룹을 포함해야 합니다.

대상 그룹 생성

1. VPC-2 내 타겟팅하려는 서비스의 IP 주소를 식별하세요.
   - 이러한 IP 주소는 피어링 연결에 사용되는 서브넷의 구성원이어야 합니다.
   - 대상 IP 주소는 고정적이고 변경할 수 없어야 합니다. 서비스에 동적 IP 주소가 있는 경우 정적 리소스(예: Network Load Balancer)를 타겟팅하고 해당 정적 리소스가 실제 대상으로 요청을 라우팅하는 것을 고려해 보세요.

> **Note**
>
> - 하나 이상의 독립형 Amazon Elastic Compute Cloud(Amazon EC2) 인스턴스를 대상으로 하는 경우, https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 열고 인스턴스를 선택하세요.
> - Amazon EC2 인스턴스의 Amazon EC2 오토 스케일링 그룹을 대상으로 하는 경우 그룹을 Network Load Balancer에 연결해야 합니다. 자세한 내용은 Amazon EC2 오토 스케일링 사용 설명서에서 로드 밸런서를 오토 스케일링 그룹에 연결을 참조하세요.
>
>   그런 다음 https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 열고 네트워크 인터페이스를 선택할 수 있습니다. 여기에서 각 가용 영역에 있는 Network Load Balancer의 각 네트워크 인터페이스에 대한 IP 주소를 볼 수 있습니다.

2. VPC-1 내에 타겟 그룹을 생성 자세한 정보는 Network Load Balancer 사용 설명서의 Network Load Balancer 대상 그룹 생성을 참조하세요.

   다른 VPC에 있는 서비스의 대상 그룹에는 다음과 같은 구성이 필요합니다.

   - 대상 유형 선택에서 IP 주소를 선택하세요.
   - VPC의 경우 로드 밸런서를 호스팅할 VPC를 선택하세요. 주제 예시에서는 VPC-1입니다.
   - 대상 등록 페이지에서 VPC-2 내 각 IP 주소의 대상을 등록하세요.

     네트워크에서 기타 프라이빗 IP 주소를 선택하세요.

     가용 영역의 경우 VPC-1 내에서 원하는 영역을 선택하세요.

     IPv4 주소의 경우 VPC-2 IP 주소를 선택하세요.

     포트의 경우 사용하고 있는 포트를 선택하세요.
   - 아래에서 보류 중인 것으로 포함을 선택하세요. 주소 지정을 마치면 보류 중인 대상 등록을 선택하세요.

이 항목의 크로스 리전 시나리오와 Network Load Balancer 사용 설명서를 사용하여 대상 그룹 구성에는 다음 값이 사용됩니다.

대상 유형

```
IP addresses
```

대상 그룹 이름

```
my-target-group
```

프로토콜/포트

```
TCP : 80
```

VPC

```
vpc-1234567890abcdefg (VPC-1)
```

네트워크

```
Other private IP address
```

가용 영역

```
all
```

IPv4 주소

```
172.16.100.60
```

포트

```
80
```

## 4단계: 새 Network Load Balancer 생성

3단계에서 설명한 대상 그룹을 사용하여 Network Load Balancer를 생성하세요. 이 작업을 수행하려면 Network Load Balancer 생성을 참조하세요.

이 항목의 크로스 리전 시나리오를 사용하면 Network Load Balancer 구성 예제에 다음 값이 사용됩니다.

로드 밸런서 이름

```
my-nlb
```

스킴

```
Internal
```

VPC

```
vpc-1234567890abcdefg (VPC-1)
```

매핑

```
us-west-2a - subnet-4i23iuufkdiufsloi

us-west-2b - subnet-7x989pkjj78nmn23j

us-west-2c - subnet-0231ndmas12bnnsds
```

프로토콜/포트

```
TCP : 80
```

대상 그룹

```
my-target-group
```

## 5단계: VPC 엔드포인트 서비스 생성

Network Load Balancer를 사용하여 VPC 엔드포인트 서비스를 생성할 수 있습니다. 이 VPC 엔드포인트 서비스를 통해 Device Farm은 인터넷 게이트웨이, NAT 인스턴스, VPN 연결과 같은 추가 인프라 없이 VPC-2 환경에서 서비스에 연결할 수 있습니다.

이 작업을 수행하려면 Amazon VPC 엔드포인트 서비스 생성을 참조하세요.

## 6단계: Device Farm에서 VPC 엔드포인트 구성 생성

이제 VPC와 Device Farm 간에 프라이빗 연결을 설정할 수 있습니다. Device Farm을 사용하면 공용 인터넷을 통해 프라이빗 서비스를 노출시키지 않고 테스트할 수 있습니다. 이 작업을 수행하려면 Device Farm에서 VPC 엔드포인트 구성 생성을 참조하십시오.

이 주제의 크로스 리전 시나리오를 사용하면 VPC 엔드포인트 구성 예제에 다음 값이 사용됩니다.

이름

```
My VPCE Configuration
```

VPCE 서비스 이름

```
com.amazonaws.vpce.us-west-2.vpce-svc-1234567890abcdefg
```

서비스 DNS 이름

```
devicefarm.com
```

## 7단계: 테스트 실행 생성

6단계에서 설명한 VPC 엔드포인트 구성을 사용하여 테스트 실행을 생성할 수 있습니다. 자세한 내용은 Device Farm에서 테스트 실행 생성 또는 세션 생성 단원을 참조하세요.

## Transit Gateway를 사용하여 확장 가능한 네트워크 만들기

두 개 이상의 VPC 작업을 수행하는 확장 가능한 네트워크를 생성하려면 Transit Gateway를 사용하여 VPC 및 온프레미스 네트워크를 상호 연결하는 네트워크 전송 허브 역할을 수행하도록 할 수 있습니다. Transit Gateway를 사용하도록 Device Farm과 동일한 리전의 VPC를 구성하려면 Device Farm의 Amazon VPC 엔드포인트 서비스에 따라 프라이빗 IP 주소를 기반으로 다른 리전의 리소스를 대상으로 지정하면 됩니다.

Transit Gateway를 생성하는 자세한 방법은 Amazon VPC Transit Gateways 가이드의 Transit Gateway는 무엇인가요?를 참조하세요.

# 사설 기기 종료

> **Important**
>
> 이 지침은 사설 장치 계약 해지에만 적용됩니다. 기타 모든 AWS 서비스 및 청구 문제에 대해서는 해당 제품의 해당 설명서를 참조하거나 AWS 지원팀에 문의하세요.

```
<## ## ## ### ## ### ##### ## ### (aws-devicefarm-support@amazon.com) # ## 30# #
```

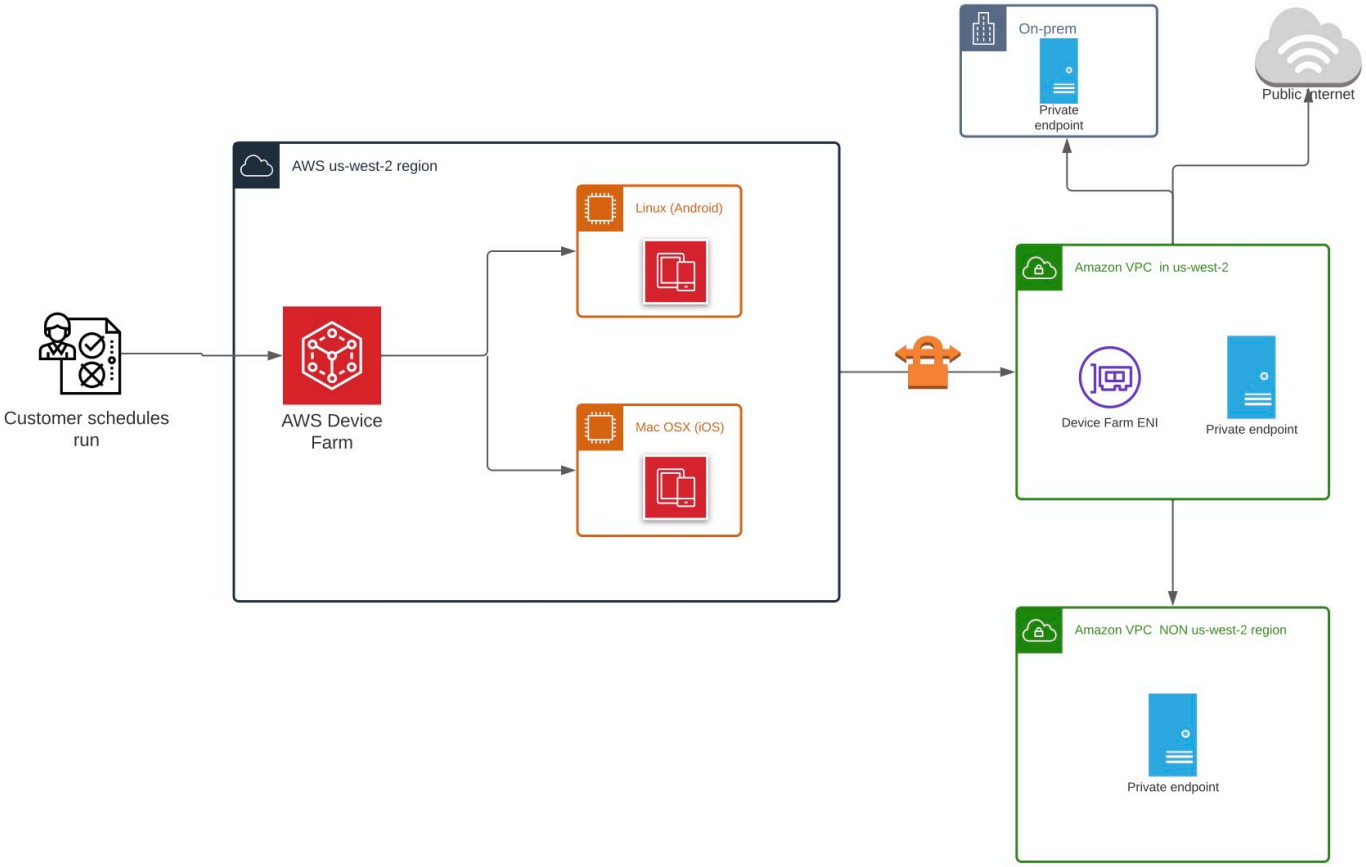
# AWS Device Farm의 VPC-ENI

> **Warning**
>
> 이 기능은 프라이빗 디바이스에서만 사용할 수 있습니다. AWS 계정에서 개인 장치 사용을 요청하려면 당사에 문의하세요. AWS 계정에 이미 프라이빗 디바이스를 추가한 경우 이 VPC 연결 방법을 사용하는 것이 좋습니다.

AWS Device Farm의 VPC-ENI 연결 기능을 통해 고객은 온 프레미스 소프트웨어 또는 다른 클라우드 공급자에 AWS호스팅된 프라이빗 엔드포인트에 안전하게 연결할 수 있습니다.

Device Farm 모바일 디바이스와 호스트 머신을 모두 해당 `us-west-2` 지역의 Amazon VPC (가상 사설 클라우드) 환경에 연결할 수 있습니다. 그러면 Elastic Network 인터페이스를 통해 격리된 non-internet-facing 서비스 및 애플리케이션에 액세스할 수 있습니다. 자세한 내용은 Amazon VPC 사용 설명서를 참조하세요.

프라이빗 엔드포인트 또는 VPC가 `us-west-2` 리전에 없는 경우 Transit Gateway 또는 VPC 피어링과 같은 솔루션을 사용하여 `us-west-2` 리전의 VPC와 연결할 수 있습니다. 이러한 경우 Device Farm은 사용자가 리전 `us-west-2` VPC용으로 제공하는 서브넷에 ENI를 생성하며, 사용자는 `us-west-2` 리전 VPC와 다른 리전의 VPC 간에 연결이 설정될 수 있도록 해야 합니다.




를 사용하여 VPC를 자동으로 생성하고 AWS CloudFormation 피어링하는 방법에 대한 자세한 내용은 의 템플릿 리포지토리에서 VPCPeering 템플릿을 참조하십시오. AWS CloudFormation GitHub

> **Note**
>
> Device Farm은 고객의 `us-west-2` VPC에서 ENI를 생성하는 데 비용을 청구하지 않습니다. 리전 간 또는 외부 VPC 간 연결 비용은 이 기능에 포함되지 않습니다.

VPC 액세스를 구성한 후에는 VPC 내에 지정된 NAT 게이트웨이가 없는 한 테스트에 사용하는 디바이스 및 호스트 머신을 VPC 외부의 리소스(예: 퍼블릭 CDN) 에 연결할 수 없습니다. 자세한 정보는 Amazon VPC 사용 설명서의 NAT 게이트웨이 단원을 참조하세요.

주제



# AWS 액세스 제어 및 IAM

AWS Device Farm을 사용하면 AWS Identity and Access Management(IAM) 을 사용하여 Device Farm의 기능에 대한 액세스를 허용하거나 제한하는 정책을 생성할 수 있습니다. AWS Device Farm에서 VPC 연결 기능을 사용하려면 AWS Device Farm에 액세스하려 사용하는 사용자 계정 또는 역할에 대해 다음과 같은 IAM 정책이 필요합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [{
      "Effect": "Allow",
      "Action": [
        "devicefarm:*",
        "ec2:DescribeVpcs",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:CreateNetworkInterface"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
      "Resource": "arn:aws:iam::*:role/aws-service-role/devicefarm.amazonaws.com/
AWSServiceRoleForDeviceFarm",
      "Condition": {
        "StringLike": {
          "iam:AWSServiceName": "devicefarm.amazonaws.com"
        }
```

```
      }
    }
  ]
}
```

VPC 구성으로 Device Farm 프로젝트를 만들거나 업데이트하려면 VPC 구성에 나열된 리소스에 대해 다음 작업을 호출할 수 있도록 IAM 정책을 사용해야 합니다.

```
"ec2:DescribeVpcs"
"ec2:DescribeSubnets"
"ec2:DescribeSecurityGroups"
"ec2:CreateNetworkInterface"
```

또한 IAM 정책에서 서비스 연결 역할 생성을 허용해야 합니다.

```
"iam:CreateServiceLinkedRole"
```

> **Note**
>
> 프로젝트에서 VPC 구성을 사용하지 않는 사용자에게는 이러한 권한이 필요하지 않습니다.

## 서비스 연결 역할

AWS Device Farm은 AWS Identity and Access Management (IAM) 서비스 연결 역할을 사용합니다. 서비스 연결 역할은 Device Farm에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 Device Farm에서 미리 정의하며 서비스가 사용자를 대신하여 다른 AWS 서비스를 호출하는 데 필요한 모든 권한을 포함합니다.

필요한 권한을 수동으로 추가할 필요가 없으므로 서비스 연결 역할은 Device Farm을 더 쉽게 설정할 수 있도록 합니다. Device Farm에서 서비스 연결 역할의 권한을 정의하므로 다르게 정의되지 않은 한, Device Farm만 해당 역할을 수임할 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함되며 이 권한 정책은 다른 IAM 엔터티에 연결할 수 없습니다.

먼저 관련 리소스를 삭제한 후에만 서비스 연결 역할을 삭제할 수 있습니다. 이렇게 하면 Device Farm 리소스에 대한 액세스 권한을 부주의로 삭제할 수 없기 때문에 리소스가 보호됩니다.

서비스 연결 역할을 지원하는 기타 서비스에 대한 자세한 내용은 IAM으로 작업하는 AWS 서비스를 참조하고 서비스 연결 역할 열에 예가 있는 서비스를 찾습니다. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예 링크를 선택합니다.

# Device Farm에 대한 서비스 연결 역할 권한

Device Farm은 Device Farm이 사용자를 AWSServiceRoleForDeviceFarm대신하여 AWS 리소스에 액세스할 수 있도록 허용이라는 서비스 연결 역할을 사용합니다.

AWSServiceRoleForDeviceFarm 서비스 연결 역할은 다음 서비스가 역할을 맡을 것으로 신뢰합니다.

- `devicefarm.amazonaws.com`

역할 권한 정책을 통해 Device Farm은 다음 작업을 완료할 수 있습니다.

- 계정의 경우
  - 네트워크 인터페이스 생성
  - 네트워크 인터페이스 설명
  - VPC 설명
  - 서브넷 설명
  - 보안 그룹 설명
  - 인터페이스 삭제
  - 네트워크 인터페이스 수정
- 네트워크 인터페이스의 경우
  - 태그 생성
- Device Farm에서 관리하는 EC2 네트워크 인터페이스의 경우
  - 네트워크 인터페이스 권한 생성

전체 IAM 정책은 다음과 같습니다.

```
{
"Version": "2012-10-17",
"Statement": [
 {
  "Effect": "Allow",
  "Action": [
   "ec2:DescribeNetworkInterfaces",
   "ec2:DescribeVpcs",
   "ec2:DescribeSubnets",
   "ec2:DescribeSecurityGroups"
  ],
```

```
    "Resource": "*"
   },
   {
    "Effect": "Allow",
    "Action": [
     "ec2:CreateNetworkInterface"
    ],
    "Resource": [
     "arn:aws:ec2:*:*:subnet/*",
     "arn:aws:ec2:*:*:security-group/*"
    ]
   },
   {
    "Effect": "Allow",
    "Action": [
     "ec2:CreateNetworkInterface"
    ],
    "Resource": [
     "arn:aws:ec2:*:*:network-interface/*"
    ],
    "Condition": {
     "StringEquals": {
      "aws:RequestTag/AWSDeviceFarmManaged": "true"
     }
    }
   },
   {
    "Effect": "Allow",
    "Action": [
     "ec2:CreateTags"
    ],
    "Resource": "arn:aws:ec2:*:*:network-interface/*",
    "Condition": {
     "StringEquals": {
      "ec2:CreateAction": "CreateNetworkInterface"
     }
    }
   },
   {
    "Effect": "Allow",
    "Action": [
     "ec2:CreateNetworkInterfacePermission",
     "ec2:DeleteNetworkInterface"
    ],
```

```
    "Resource": "arn:aws:ec2:*:*:network-interface/*",
    "Condition": {
     "StringEquals": {
      "aws:ResourceTag/AWSDeviceFarmManaged": "true"
     }
    }
   },
   {
    "Effect": "Allow",
    "Action": [
     "ec2:ModifyNetworkInterfaceAttribute"
    ],
    "Resource": [
     "arn:aws:ec2:*:*:security-group/*",
     "arn:aws:ec2:*:*:instance/*"
    ]
   },
   {
    "Effect": "Allow",
    "Action": [
     "ec2:ModifyNetworkInterfaceAttribute"
    ],
    "Resource": "arn:aws:ec2:*:*:network-interface/*",
    "Condition": {
     "StringEquals": {
      "aws:ResourceTag/AWSDeviceFarmManaged": "true"
     }
    }
   }
  ]
}
```

IAM 엔터티(사용자, 그룹, 역할 등)가 서비스 링크 역할을 생성하고 편집하거나 삭제할 수 있도록 권한을 구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 권한 단원을 참조하세요.

## Device Farm에 대한 서비스 연결 역할 생성

모바일 테스트 프로젝트에 VPC 구성을 제공할 때 서비스 연결 역할을 수동으로 생성할 필요가 없습니다. AWS Management Console AWS CLI, 또는 AWS API에서 첫 번째 Device Farm 리소스를 생성하면 Device Farm이 서비스 연결 역할을 자동으로 생성합니다.

이 서비스 연결 역할을 삭제했다가 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. Device Farm 리소스를 처음 만들 때 Device Farm은 서비스 연결 역할을 다시 생성합니다.

또한 IAM 콘솔을 사용해 Device Farm 사용 사례로 서비스 연결 역할을 생성할 수도 있습니다. AWS CLI 또는 AWS API에서 서비스 이름을 사용하여 서비스 연결 역할을 생성합니다. `devicefarm.amazonaws.com` 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 생성 단원을 참조하세요. 이 서비스 연결 역할을 삭제하면 동일한 프로세스를 사용하여 역할을 다시 생성할 수 있습니다.

## Device Farm에 대한 서비스 연결 역할 편집

Device Farm에서는 AWSServiceRoleForDeviceFarm 서비스 연결 역할을 편집할 수 없습니다. 서비스 링크 역할을 생성한 후에는 다양한 개체가 역할을 참조할 수 있기 때문에 역할 이름을 변경할 수 없습니다. 하지만 IAM을 사용하여 역할의 설명을 편집할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 편집을 참조하세요.

## Device Farm에 대한 서비스 연결 역할 삭제

서비스 연결 역할이 필요한 기능 또는 서비스가 더 이상 필요 없는 경우에는 해당 역할을 삭제하는 것이 좋습니다. 따라서 적극적으로 모니터링하거나 유지하지 않는 미사용 엔터티가 없도록 합니다. 단, 서비스 링크 역할에 대한 리소스를 먼저 정리해야 수동으로 삭제할 수 있습니다.

> **Note**
>
> 리소스를 삭제하려 할 때 Device Farm 서비스가 역할을 사용 중이면 삭제에 실패할 수 있습니다. 이 문제가 발생하면 몇 분 기다렸다가 작업을 다시 시도하세요.

IAM을 사용하여 수동으로 서비스 연결 역할을 삭제하려면

IAM 콘솔 AWS CLI, 또는 AWS API를 사용하여 서비스 연결 역할을 삭제합니다. AWSServiceRoleForDeviceFarm 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 삭제 단원을 참조하세요.

## Device Farm 서비스 연결 역할이 지원되는 리전

Device Farm는 서비스가 제공되는 모든 리전에서 서비스 연결 역할을 사용하도록 지원합니다. 자세한 내용은 AWS 리전 및 엔드포인트 단원을 참조하세요.

에서는 서비스가 제공되는 모든 리전에서 서비스 연결 역할을 사용하도록 지원하지 않습니다. 다음 지역에서 AWSServiceRoleForDeviceFarm 역할을 사용할 수 있습니다.

| 지역명 | 리전 자격 증명 | Device Farm 지원 |
|---|---|---|
| 미국 동부(버지니아 북부) | us-east-1 | 아니요 |
| 미국 동부(오하이오) | us-east-2 | 아니요 |
| 미국 서부(캘리포니아 북부) | us-west-1 | 아니요 |
| 미국 서부(오레곤) | us-west-2 | 예 |
| 아시아 태평양(뭄바이) | ap-south-1 | 아니요 |
| 아시아 태평양(오사카) | ap-northeast-3 | 아니요 |
| 아시아 태평양(서울) | ap-northeast-2 | 아니요 |
| 아시아 태평양(싱가포르) | ap-southeast-1 | 아니요 |
| 아시아 태평양(시드니) | ap-southeast-2 | 아니요 |
| 아시아 태평양(도쿄) | ap-northeast-1 | 아니요 |
| 캐나다(중부) | ca-central-1 | 아니요 |
| 유럽(프랑크푸르트) | eu-central-1 | 아니요 |
| 유럽(아일랜드) | eu-west-1 | 아니요 |
| 유럽(런던) | eu-west-2 | 아니요 |
| 유럽(파리) | eu-west-3 | 아니요 |
| 남아메리카(상파울루) | sa-east-1 | 아니요 |
| AWS GovCloud (US) | us-gov-west-1 | 아니요 |

# 사전 조건

다음 목록은 VPC-ENI 구성을 만들 때 검토해야 할 몇 가지 요구 사항 및 제안 사항을 설명합니다.

- AWS 계정에 사설 장치를 할당해야 합니다.
- 서비스 연결 역할을 생성할 권한이 있는 AWS 계정 사용자 또는 역할이 있어야 합니다. Device Farm 모바일 테스트 기능과 함께 Amazon VPC 엔드포인트를 사용하는 경우 Device Farm은 AWS Identity and Access Management (IAM) 서비스 연결 역할을 생성합니다.
- Device Farm은 해당 `us-west-2` 리전의 VPC에만 연결할 수 있습니다. `us-west-2` 리전에 VPC가 없는 경우 하나를 생성해야 합니다. 그런 다음 다른 리전의 VPC에 있는 리소스에 액세스하려면 해당 `us-west-2` 리전의 VPC와 다른 리전의 VPC 간에 피어링 연결을 설정해야 합니다. VPC 피어링에 대한 자세한 내용은 Amazon VPC 피어링 안내서를 참조하세요.

  연결을 구성할 때 지정된 VPC에 액세스할 수 있는지 확인해야 합니다. Device Farm에 대한 특정 Amazon Elastic Compute Cloud(Amazon EC2) 권한을 구성해야 합니다.
- 사용하는 VPC에는 DNS 확인이 필요합니다.
- VPC가 생성되면 해당 `us-west-2`리전의 VPC에 대한 다음 정보가 필요합니다.
  - VPC ID
  - 서브넷 ID
  - 보안 그룹 ID
- 프로젝트별로 Amazon VPC 연결을 구성해야 합니다. 지금은 프로젝트당 하나의 VPC 구성만 구성할 수 있습니다. VPC를 구성할 때 Amazon VPC는 VPC 내에 인터페이스를 생성하여 지정된 서브넷 및 보안 그룹에 할당합니다. 프로젝트와 관련된 모든 향후 세션에서는 구성된 VPC 연결을 사용합니다.
- VPC-ENI 구성은 기존 VPCE 기능과 함께 사용할 수 없습니다.
- 기존 프로젝트에는 실행 수준에서 유지되는 VPCE 설정이 있을 수 있으므로 VPC-ENI 구성으로 기존 프로젝트를 업데이트하지 않는 것이 좋습니다. 대신 기존 VPCE 기능을 이미 사용하고 있다면 모든 새 프로젝트에 VPC-ENI를 사용하세요.

# Amazon VPC에 연결

Amazon VPC 엔드포인트를 사용하도록 프로젝트를 구성하고 업데이트할 수 있습니다. VPC-ENI 구성은 프로젝트별로 구성됩니다. 프로젝트는 한 번에 VPC-ENI 엔드포인트를 하나만 가질 수 있습니다. 프로젝트에 VPC 액세스를 구성하려면 다음 세부 정보를 알아야 합니다.

- 앱이 호스팅되는 경우의 us-west-2에서 VPC ID 또는 다른 리전의 다른 VPC에 연결되는 us-west-2 VPC ID
- 연결에 적용할 수 있는 보안 그룹
- 연결과 연결될 서브넷 세션이 시작되면 사용 가능한 가장 큰 서브넷이 사용됩니다. VPC 연결의 가용성 상태를 개선하려면 여러 가용 영역에 여러 서브넷을 연결하는 것이 좋습니다.

VPC-ENI 구성을 생성한 후에는 아래 단계에 따라 콘솔 또는 CLI를 사용하여 세부 정보를 업데이트할 수 있습니다.

Console

1. https://console.aws.amazon.com/devicefarm에서 Device Farm 콘솔에 로그인하세요.
2. Device Farm 탐색 패널에서 모바일 디바이스 테스트를 선택한 다음 프로젝트를 선택하세요.
3. 모바일 테스트 프로젝트의 목록에서 프로젝트 이름을 선택하세요.
4. 프로젝트 설정을 선택하세요.
5. Virtual Private Cloud(VPC) 설정 단원에서 VPC, Subnets 및 Security Groups를 변경할 수 있습니다.
6. 저장을 선택하세요.

CLI

다음 AWS CLI 명령을 사용하여 Amazon VPC를 업데이트하세요.

```
$  aws devicefarm update-project \
--arn arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef \
--vpc-config \
securityGroupIds=sg-02c1537701a7e3763,sg-005dadf9311efda25,\
subnetIds=subnet-09b1a45f9cac53717,subnet-09b1a45f9cac12345,\
vpcId=vpc-0238fb322af81a368
```

프로젝트를 생성할 때 Amazon VPC를 구성할 수도 있습니다.

```
$  aws devicefarm create-project \
--name VPCDemo \
--vpc-config \
```

```
securityGroupIds=sg-02c1537701a7e3763,sg-005dadf9311efda25,\
subnetIds=subnet-09b1a45f9cac53717,subnet-09b1a45f9cac12345,\
vpcId=vpc-0238fb322af81a368
```

## Limits

VPC-ENI 기능에는 다음과 같은 제한이 적용됩니다.

- Device Farm 프로젝트의 VPC 구성에는 최대 5개의 보안 그룹을 제공할 수 있습니다.
- Device Farm 프로젝트의 VPC 구성에는 최대 8개의 서브넷을 제공할 수 있습니다.
- VPC와 함께 작동하도록 Device Farm 프로젝트를 구성할 때 제공할 수 있는 가장 작은 서브넷에는 사용 가능한 IPv4 주소가 5개 이상 있어야 합니다.
- 퍼블릭 IP 주소는 현재 지원되지 않습니다. 대신 Device Farm 프로젝트에서 프라이빗 서브넷을 사용하는 것이 좋습니다. 테스트 중에 공용 인터넷 액세스가 필요한 경우 Network Address Translation(NAT) 게이트웨이를 사용하세요. Device Farm 프로젝트를 퍼블릭 서브넷으로 구성해도 테스트에 인터넷 액세스 권한 또는 퍼블릭 IP 주소는 제공되지 않습니다.
- 서비스 관리형 ENI에서 나가는 트래픽만 지원됩니다. 즉, ENI는 VPC로부터 요청하지 않은 인바운드 요청을 수신할 수 없습니다.

# Device Farm - 레거시와 함께 Amazon VPC 엔드포인트 서비스 사용 (권장되지 않음)

**Warning**

VPCE는 이제 레거시 기능으로 간주되므로 프라이빗 엔드포인트 연결에는 이 페이지에 설명된 VPC-ENI 연결을 사용하는 것이 좋습니다. VPC-ENI는 VPCE 연결 방법에 비해 더 유연하고, 더 간단한 구성을 제공하며, 더 비용 효율적이며, 유지 관리 오버헤드가 훨씬 적습니다.

**Note**

Device Farm과 함께 Amazon VPC 엔드포인트 서비스를 사용하는 것은 프라이빗 디바이스가 구성된 고객만 가능합니다. AWS 계정에서 프라이빗 디바이스와 함께 이 기능을 사용할 수 있도록 하려면 문의하기하세요.

Amazon VPC (가상 사설 클라우드) 는 사용자가 정의한 가상 네트워크에서 AWS 리소스를 시작하는 데 사용할 수 있는 AWS 서비스입니다. VPC를 사용하여 IP 주소 범위, 서브넷, 라우팅 테이블, 네트워크 게이트웨이 등의 네트워크 설정을 제어할 수 있습니다.

Amazon VPC를 사용하여 미국 서부 (오레곤) (`us-west-2`) AWS 지역에서 프라이빗 애플리케이션을 호스팅하는 경우, VPC와 Device Farm 사이에 프라이빗 연결을 설정할 수 있습니다. 이 연결이 있으면 Device Farm을 사용하여 공용 인터넷에 노출되지 않고 프라이빗 애플리케이션을 테스트할 수 있습니다. AWS 계정에서 개인 디바이스에서 이 기능을 사용할 수 있도록 하려면 당사에 문의하십시오.

VPC의 리소스를 Device Farm에 연결하려면 Amazon VPC 콘솔을 사용하여 VPC 엔드포인트 서비스를 생성할 수 있습니다. 이 엔드포인트 서비스를 사용하면 Device Farm VPC 엔드포인트를 통해 VPC의 리소스를 Device Farm에 제공할 수 있습니다. 이 엔드포인트를 이용하면 인터넷 게이트웨이나 네트워크 주소 변환(NAT) 인스턴스 또는 VPN 연결 없이도 Device Farm에 안정적이고 확장 가능하게 연결됩니다. 자세한 내용은 가이드의 VPC 엔드포인트 서비스 PrivateLink (AWS) 를 참조하십시오.AWS PrivateLink

> **Important**
>
> Device Farm VPC 엔드포인트 기능을 사용하면 연결을 사용하여 VPC의 프라이빗 내부 서비스를 Device Farm 퍼블릭 VPC에 안전하게 연결할 수 있습니다. AWS PrivateLink 연결이 안전하고 비공개이긴 하지만, 보안은 AWS 보안 인증 정보 보호에 따라 달라집니다. AWS 자격 증명이 침해되는 경우 공격자는 서비스 데이터에 액세스하거나 서비스 데이터를 외부에 노출시킬 수 있습니다.

Amazon VPC에서 VPC 엔드포인트 서비스를 생성한 후, Device Farm 콘솔을 사용하여 Device Farm에서 VPC 엔드포인트 구성을 생성할 수 있습니다. 여기에서는 Device Farm에서 Amazon VPC 연결 및 VPC 엔드포인트 구성을 생성하는 방법을 보여줍니다.

## 시작하기 전 준비 사항

다음 정보는 us-west-2a, us-west-2b, us-west-2c와 같은 각 가용 영역 내 서브넷을 가진 미국 서부(오레곤)(`us-west-2`) 리전의 Amazon VPC 사용자를 위한 것입니다.

Device Farm에는 함께 사용할 수 있는 VPC 엔드포인트 서비스에 대한 추가 요구 사항이 있습니다. Device Farm과 함께 작동하도록 VPC 엔드포인트 서비스를 생성하고 구성할 때는 다음 요구 사항을 충족하는 옵션을 선택해야 합니다.

- 서비스의 가용 영역에는 us-west-2a, us-west-2b, us-west-2c가 포함되어야 합니다. VPC 엔드포인트 서비스와 연결된 Network Load Balancer는 해당 VPC 엔드포인트 서비스의 가용 영역을 결정합니다. VPC 엔드포인트 서비스에 이러한 가용 영역 3개가 모두 표시되지 않는 경우 Network Load Balancer를 다시 생성하여 이 세 영역을 활성화한 다음 Network Load Balancer를 엔드포인트 서비스에 다시 연결해야 합니다.
- 엔드포인트 서비스에 허용된 보안 주체에는 Device Farm VPC 엔드포인트(서비스 ARN)의 Amazon 리소스 이름(ARN)이 포함되어야 합니다. 엔드포인트 서비스를 생성한 후 Device Farm VPC 엔드포인트 서비스 ARN을 허용 목록에 추가하여 Device Farm에 VPC 엔드포인트 서비스에 액세스할 수 있는 권한을 부여합니다. Device Farm VPC 엔드포인트 서비스 ARN을 받으려면 문의하기하세요.

또한 VPC 엔드포인트 서비스를 생성할 때 수락 필요 설정을 계속 켜두면 Device Farm이 엔드포인트 서비스에 보내는 각 연결 요청을 수동으로 수락해야 합니다. 기존 엔드포인트 서비스에서 이 설정을 변경하려면 Amazon VPC 콘솔에서 엔드포인트 서비스를 선택하고 작업을 선택한 다음 엔드포인트 수락 설정 수정을 선택하세요. 자세한 내용은AWS PrivateLink 가이드 내 로드 밸런서 및 수락 설정 변경을 참조하세요.

다음 단원에서는 이러한 요구 사항을 충족하는 Amazon VPC 엔드포인트 서비스를 생성하는 방법을 설명합니다.

## 1단계: Network Load Balancer 생성

VPC와 Device Farm 간에 프라이빗 연결을 설정하는 첫 번째 단계는 요청을 대상 그룹으로 라우팅하는 Network Load Balancer를 생성하는 것입니다.

New console

새 콘솔을 사용하여 Network Load Balancer 생성

1. https://console.aws.amazon.com/ec2/에서 Amazon Elastic Compute Cloud(Amazon EC2) 콘솔을 여세요.
2. 탐색 창의 로드 밸런싱에서 로드 밸런서를 선택하세요.
3. 로드 밸런서 생성을 선택하세요.
4. Network Load Balancer에서 생성을 선택하세요.
5. Network Load Balancer 생성 페이지의 기본 구성에서 다음을 수행하세요.
   a. 로드 밸런서 이름을 입력하세요.
   b. 스킴 에서 내부를 선택하세요.

6. 네트워크 매핑에서 다음을 수행하세요.

   a. 대상 그룹의 VPC를 선택하세요.

   b. 다음과 같은 매핑을 선택하세요.

   - `us-west-2a`
   - `us-west-2b`
   - `us-west-2c`

7. 리스너 및 라우팅에서 프로토콜 및 포트 옵션을 사용하여 대상 그룹을 선택하세요.

> **Note**
>
> 교차 가용성 영역 로드 밸런싱은 기본적으로 비활성화되어 있습니다.
> 로드 밸런서는 가용 영역 `us-west-2a`, `us-west-2b`, `us-west-2c`을 사용하기 때문에 각 가용 영역에 대상을 등록해야 하거나, 3개 영역 미만의 영역에 대상을 등록하는 경우 교차 영역 로드 밸런싱을 활성화해야 합니다. 그렇지 않으면 로드 밸런서는 예상대로 작동하지 않을 수 있습니다.

8. 로드 밸런서 생성을 선택하세요.

Old console

이전 콘솔을 사용하여 Network Load Balancer를 생성

1. https://console.aws.amazon.com/ec2/에서 Amazon Elastic Compute Cloud(Amazon EC2) 콘솔을 여세요.
2. 탐색 창의 로드 밸런싱에서 로드 밸런서를 선택하세요.
3. 로드 밸런서 생성을 선택하세요.
4. Network Load Balancer에서 생성을 선택하세요.
5. 로드 밸런서 구성 페이지의 기본 구성에서 다음을 수행하세요.

   a. 로드 밸런서 이름을 입력하세요.

   b. 스킴에서 내부를 선택하세요.

6. 리스너에서 대상 그룹이 사용하는 프로토콜과 포트를 선택하세요.
7. 가용 영역에서 다음을 수행하세요.

   a. 대상 그룹의 VPC를 선택하세요.

b. 다음과 같은 가용 영역을 선택하세요.

- us-west-2a
- us-west-2b
- us-west-2c

c. 다음: 보안 설정 구성을 선택하세요.

8. (선택 사항) 보안 설정을 구성한 후 다음: 라우팅 구성을 선택하세요.
9. 라우팅 구성 페이지에서 다음을 수행하세요.

a. 대상 그룹에서 기존 대상 그룹을 선택하세요.

b. 이름에서 대상 그룹을 선택하세요.

c. 다음: 대상 등록을 선택하세요.

10. 대상 등록 페이지에서 대상을 검토한 후 다음: 검토를 선택하세요.

> **Note**
>
> 교차 가용성 영역 로드 밸런싱은 기본적으로 비활성화되어 있습니다. 로드 밸런서는 가용 영역 us-west-2a, us-west-2b, us-west-2c을 사용하기 때문에 각 가용 영역에 대상을 등록해야 하거나, 3개 영역 미만의 영역에 대상을 등록하는 경우 교차 영역 로드 밸런싱을 활성화해야 합니다. 그렇지 않으면 로드 밸런서는 예상대로 작동하지 않을 수 있습니다.

11. 로드 밸런서 구성을 검토하고 생성을 선택하세요.

## 2단계: Amazon VPC 엔드포인트 생성

Network Load Balancer를 생성한 후 Amazon VPC 콘솔을 사용하여 VPC에 엔드포인트 서비스를 생성하세요.

1. https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 여세요.
2. 리전별 리소스에서 엔드포인트 서비스를 선택하세요.
3. 엔드포인트 서비스 생성을 선택하세요.
4. 다음 중 하나를 수행하십시오.

- 엔드포인트 서비스에서 사용할 Network Load Balancer가 이미 있는 경우 사용 가능한 로드 밸런서에서 해당 Network Load Balancer를 선택한 다음 5단계로 넘어가세요.

- Network Load Balancer를 아직 생성하지 않았다면 새 로드 밸런서 생성을 선택하세요. Amazon EC2 콘솔을 여세요. 3단계부터 시작하여 Network Load Balancer 생성의 단계를 수행한 다음 Amazon VPC 콘솔에서 이 단계를 계속 진행하세요.

5. 포함된 가용 영역의 경우 목록에 `us-west-2a`, `us-west-2b`, `us-west-2c`가 나타나는지 확인하세요.
6. 엔드포인트 서비스로 전송되는 각 연결 요청을 수동으로 수락하거나 거부하고 싶지 않다면 추가 설정에서 수락 필요를 선택 해제하세요. 이 확인란의 선택을 취소하면 엔드포인트 서비스가 수신하는 각 연결 요청을 자동으로 수락합니다.
7. 생성을 선택하세요.
8. 새 엔드포인트 서비스에서 주체 허용을 선택하세요.
9. 엔드포인트 서비스의 허용 목록에 추가할 Device Farm VPC 엔드포인트(서비스 ARN)의 ARN을 받으려면 문의하기 후, 해당 서비스 ARN을 서비스의 허용 목록에 추가하세요.
10. 엔드포인트 서비스의 세부 정보 탭에서 서비스의 이름(서비스 이름)을 기록해 두세요. 이 이름은 다음 단계에서 VPC 엔드포인트 구성을 생성할 때 필요합니다.

이제 VPC 엔드포인트 서비스를 Device Farm과 사용할 준비를 마쳤습니다.

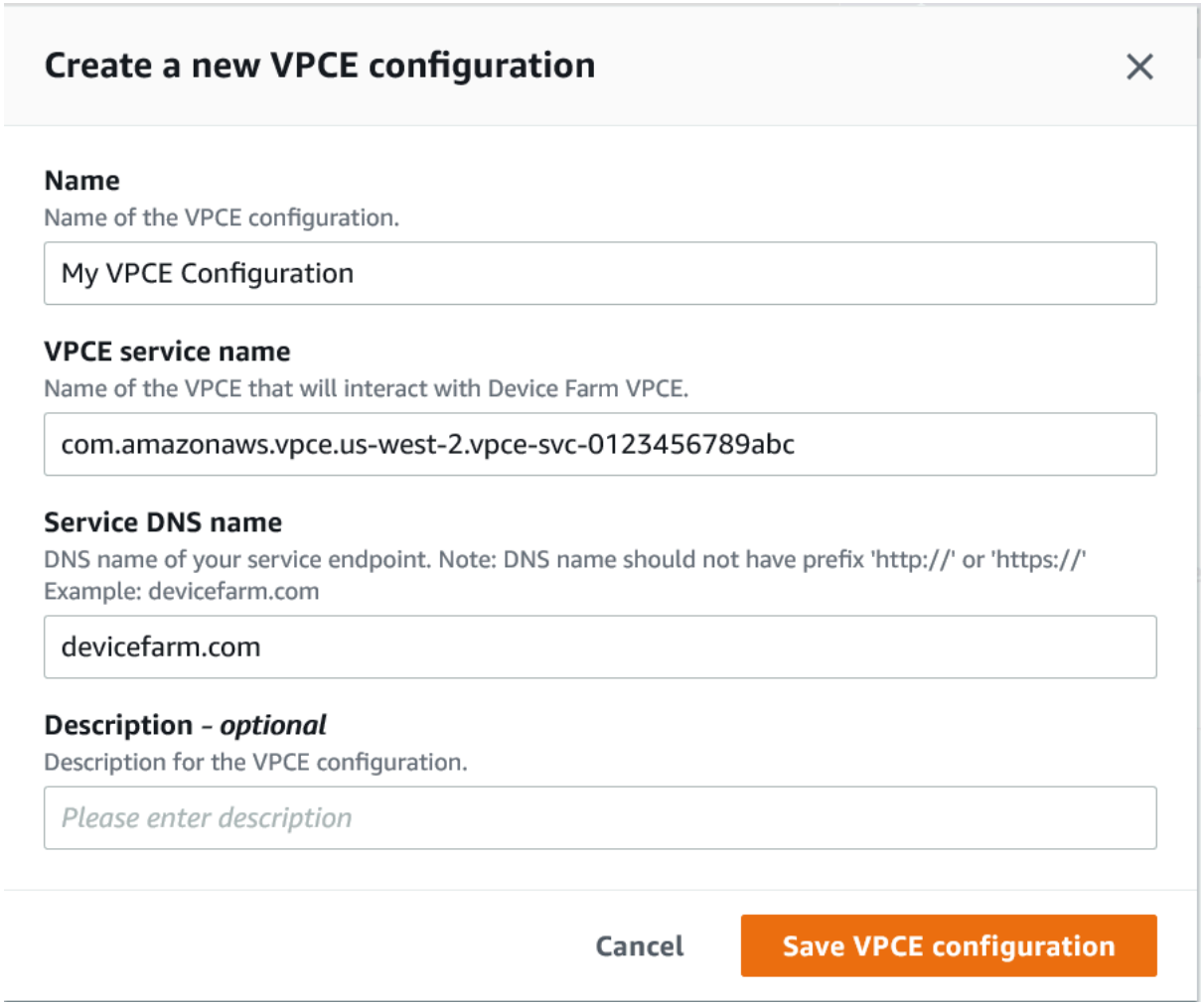
## 3단계: Device Farm에서 VPC 엔드포인트 구성 생성

Amazon VPC에서 엔드포인트 서비스를 생성한 후, Device Farm에서 Amazon VPC 엔드포인트 구성을 생성할 수 있습니다.

1. https://console.aws.amazon.com/devicefarm에서 Device Farm 콘솔에 로그인하세요.
2. 탐색 창에서 모바일 디바이스 테스트를 선택하고 프라이빗 디바이스를 선택하세요.
3. VPCE 구성을 선택하세요.
4. VPCE 구성 생성을 선택하세요.
5. 새 VPCE 구성 생성에서 VPC 엔드포인트 구성의 이름을 입력하세요.
6. VPCE 서비스 이름에 Amazon VPC 콘솔에서 기록해 둔 Amazon VPC 엔드포인트 서비스(서비스 이름)를 입력하세요. 이름은 `com.amazonaws.vpce.us-west-2.vpce-svc-id`와 같은 형식입니다.
7. 서비스 DNS 이름에 테스트하려는 앱의 서비스 DN 이름(예: `devicefarm.com`)을 입력하세요. 서비스 DNS 이름 앞에 `http` 또는 `https`를 지정하지 마세요.

공용 인터넷을 통해서는 도메인 이름에 액세스할 수 없습니다. 또한 VPC 엔드포인트 서비스에 매핑되는 이 새 도메인 이름은 Amazon Route 53에서 생성되며 Device Farm 세션에서만 독점적으로 사용할 수 있습니다.

8. 저장을 선택하세요.




## 4단계: 테스트 실행 생성

VPC 엔드포인트 구성을 저장한 후 구성을 사용하여 테스트 실행을 생성하거나 원격 액세스 세션을 생성할 수 있습니다. 자세한 내용은 Device Farm에서 테스트 실행 생성 또는 세션 생성을 참조하세요.

# AWS CloudTrail을 사용하여 AWS Device Farm API 호출 로깅

AWS Device Farm은 사용자, 역할 또는 AWS Device Farm 내 AWS 서비스가 수행한 작업에 대한 레코드를 제공하는 서비스인 AWS CloudTrail와 통합됩니다. CloudTrail은 AWS Device Farm에 대한 모든 API 호출을 이벤트로 캡처합니다. 캡처되는 호출에는 AWS Device Farm 콘솔에서 수행한 호출과 AWS Device Farm API 작업에 대한 코드 호출이 포함됩니다. 추적을 생성하면 AWS Device Farm 이벤트를 포함한 CloudTrail 이벤트를 지속적으로 Amazon S3 버킷에 배포할 수 있습니다. 추적을 구성하지 않은 경우에도 CloudTrail 콘솔의 이벤트 기록에서 최신 이벤트를 볼 수 있습니다. CloudTrail에서 수집한 정보를 사용하여 AWS Device Farm에 수행된 요청, 요청이 수행된 IP 주소, 수행한 사람, 수행 시간 및 추가 세부 정보를 확인할 수 있습니다.

CloudTrail에 대한 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하세요.

## CloudTrail의 AWS Device Farm 정보

CloudTrail은 계정 생성 시 AWS 계정에서 사용되도록 설정됩니다. AWS Device Farm에서 활동이 발생하면 해당 활동이 이벤트 기록의 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. AWS 계정에서 최신 이벤트를 확인, 검색 및 다운로드할 수 있습니다. 자세한 내용은 CloudTrail 이벤트 기록을 사용하여 이벤트 보기를 참조하세요.

AWS Device Farm의 이벤트를 포함하여 AWS 계정에 이벤트를 지속적으로 기록하려면 추적을 생성하세요. CloudTrail은 추적을 사용하여 Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 추적을 생성하면 기본적으로 모든 AWS 리전에 추적이 적용됩니다. 추적은 AWS 파티션에 있는 모든 리전의 이벤트를 로깅하고 지정된 Amazon S3 버킷으로 로그 파일을 전송합니다. 추가적으로, CloudTrail 로그에서 수집된 이벤트 데이터를 추가 분석 및 처리하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 다음 자료를 참조하세요.

- 추적 생성 개요
- CloudTrail 지원 서비스 및 통합
- CloudTrail에 대한 Amazon SNS 알림 구성
- 여러 리전에서 CloudTrail 로그 파일 받기 및 여러 계정에서 CloudTrail 로그 파일 받기

AWS 계정에서 CloudTrail 로깅을 사용하는 경우 Device Farm 작업에 대한 API 호출이 로그 파일에서 추적됩니다. Device Farm 레코드가 다른 AWS 서비스 레코드와 함께 로그 파일에 기록됩니다. CloudTrail은 기간 및 파일 크기를 기준으로 새 파일을 만들고 기록하는 시점을 결정합니다.

모든 Device Farm 작업은 AWS CLI 참조 및 Device Farm 자동화에서 로깅되고 기록됩니다. 예를 들어 Device Farm에서 새 프로젝트를 생성하도록 호출하면 CloudTrail 로그 파일에 항목이 생성됩니다.

모든 이벤트 및 로그 항목에는 요청을 생성한 사용자에 대한 정보가 들어 있습니다. 신원 정보를 이용하면 다음을 쉽게 알아볼 수 있습니다.

- 루트로 요청되었는지 아니면 AWS Identity and Access Management(IAM) 사용자 자격 증명으로 했는지.
- 역할 또는 연동 사용자를 위한 임시 보안 인증으로 요청을 생성하였는지.
- 다른 AWS 서비스에서 요청했는지.

자세한 내용은 CloudTrail userIdentity 요소를 참조하세요.

## AWS Device Farm 로그 파일 항목 이해

추적이란 지정한 Amazon S3 버킷에 이벤트를 로그 파일로 입력할 수 있게 하는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함될 수 있습니다. 이벤트는 모든 소스의 단일 요청을 나타내며 요청된 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보를 포함합니다. CloudTrail 로그 파일은 퍼블릭 API 호출의 주문 스택 추적이 아니므로 특정 순서로 표시되지 않습니다.

다음은 Device Farm `ListRuns` 태스크를 보여 주는 CloudTrail 로그 항목이 나타낸 예제입니다.

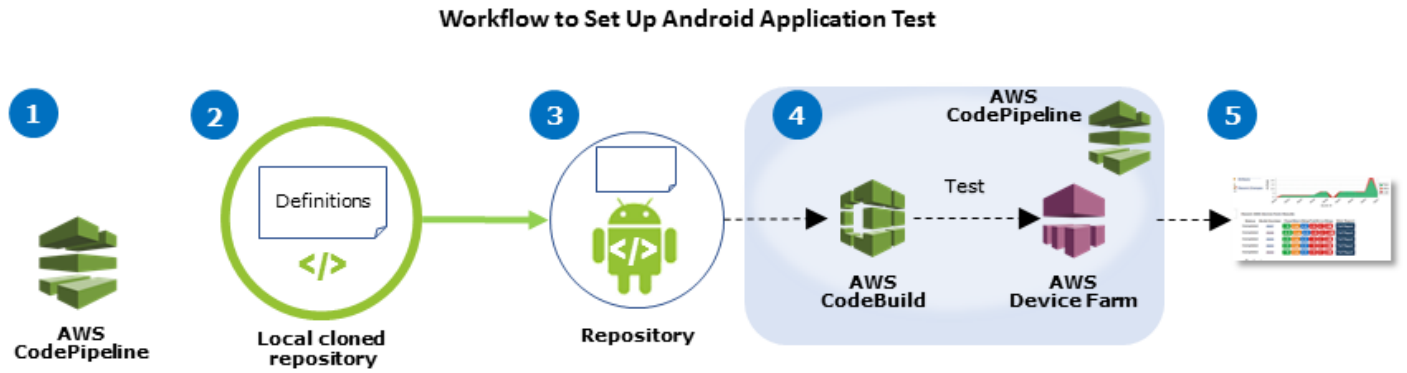
```
{
  "Records": [
    {
      "eventVersion": "1.03",
      "userIdentity": {
        "type": "Root",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:root",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2015-07-08T21:13:35Z"
          }
        }
      },
      "eventTime":"2015-07-09T00:51:22Z",
```

```
      "eventSource": "devicefarm.amazonaws.com",
      "eventName":"ListRuns",
      "awsRegion":"us-west-2",
      "sourceIPAddress":"203.0.113.11",
      "userAgent":"example-user-agent-string",
      "requestParameters": {
        "arn":"arn:aws:devicefarm:us-west-2:123456789012:project:a9129b8c-
df6b-4cdd-8009-40a25EXAMPLE"},
        "responseElements": {
          "runs": [
            {
              "created": "Jul 8, 2015 11:26:12 PM",
              "name": "example.apk",
              "completedJobs": 2,
              "arn": "arn:aws:devicefarm:us-west-2:123456789012:run:a9129b8c-
df6b-4cdd-8009-40a256aEXAMPLE/1452d105-e354-4e53-99d8-6c993EXAMPLE",
              "counters": {
                "stopped": 0,
                "warned": 0,
                "failed": 0,
                "passed": 4,
                "skipped": 0,
                "total": 4,
                "errored": 0
              },
              "type": "BUILTIN_FUZZ",
              "status": "RUNNING",
              "totalJobs": 3,
              "platform": "ANDROID_APP",
              "result": "PENDING"
            },
            ... additional entries ...
          ]
        }
      }
    }
  ]
}
```

# CodePipeline 테스트 단계에서 AWS Device Farm 사용

AWS CodePipeline을 사용하여 Device Farm에 구성된 모바일 앱 테스트를 AWS에서 관리하는 자동 릴리스 파이프라인에 통합할 수 있습니다. 필요나 일정에 따라 또는 지속적 통합 흐름의 일부로 테스트를 실행하도록 파이프라인을 구성할 수 있습니다.

다음 다이어그램은 푸시가 리포지토리에 커밋될 때마다 Android 앱이 빌드되고 테스트되는 지속적 통합 흐름을 보여줍니다. 이 파이프라인 구성을 생성하려면 튜토리얼: GitHub에 푸시된 Android 앱 빌드 및 테스트를 참조하세요.




| 1. 구성 | 2. 정의 추가 | 3. 푸시 | 4. 빌드 및 테스트 | 5. 보고서 |
|---|---|---|---|---|
| 파이프라인 리소스 구성 | 패키지에 빌드 및 테스트 정의 추가 | 리포지토리에 패키지 푸시 | 자동으로 시작된 빌드 출력 아티팩트를 빌드하고 테스트하는 앱 | 테스트 결과 보기 |

컴파일된 앱(예: iOS `.ipa` 또는 Android `.apk` 파일)을 소스로 지속적으로 테스트하는 파이프라인을 구성하는 방법을 알아보려면 튜토리얼: .ipa 파일을 Amazon S3 버킷에 업로드할 때마다 iOS 앱 테스트를 참조하세요.

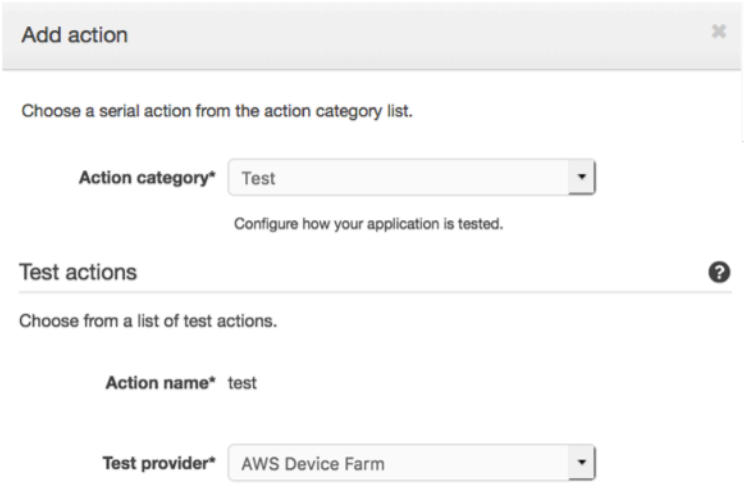
## Device Farm 테스트를 사용하도록 CodePipeline 구성

이 단계에서는 Device Farm 프로젝트를 구성했으며 파이프라인을 생성했다고 가정합니다. 테스트 정의와 컴파일된 앱 패키지 파일이 포함된 입력 아티팩트를 수신하는 테스트 단계로 파이프라인을 구성

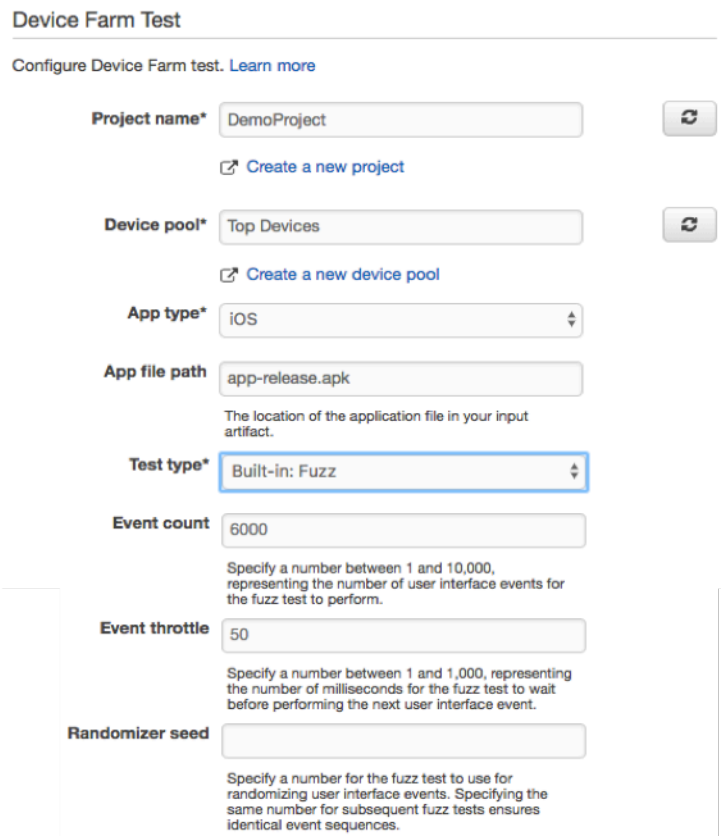
해야 합니다. 테스트 스테이지 입력 아티팩트는 파이프라인에 구성된 소스 또는 빌드 스테이지의 출력 아티팩트일 수 있습니다.

Device Farm 테스트 실행을 CodePipeline 테스트 동작으로 구성

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/codepipeline/에서 CodePipeline 콘솔을 여세요.
2. 앱 출시에 사용할 파이프라인을 선택하세요.
3. 테스트 단계 패널에서 연필 아이콘을 선택한 다음 작업을 선택하세요.
4. 액션 추가 패널의 액션 카테고리에서 테스트를 선택하세요.
5. 작업 이름에 이름을 입력하세요.
6. 작업 공급자에서 AWS Device Farm을 선택하세요.

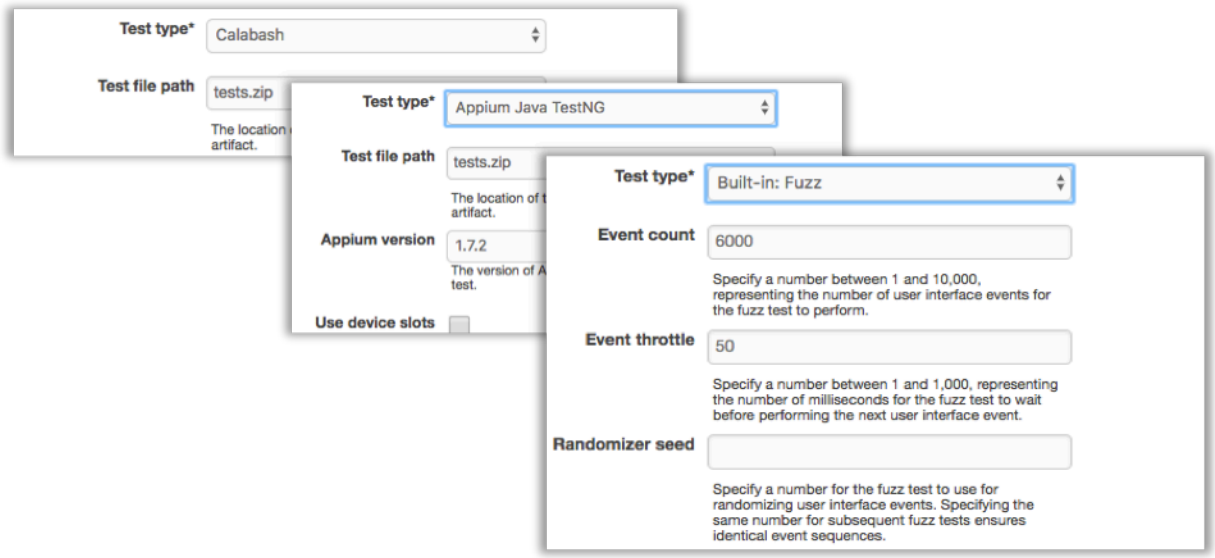



7. 프로젝트 이름에서 기존 Device Farm 프로젝트를 선택하거나 새 프로젝트 생성을 선택하세요.
8. 디바이스 풀에서 기존 디바이스 풀을 선택하거나 새 디바이스 풀 생성을 선택하세요. 디바이스 풀을 생성하는 경우 테스트 디바이스 세트를 선택해야 합니다.
9. 앱 유형에서 앱의 플랫폼을 선택하세요.

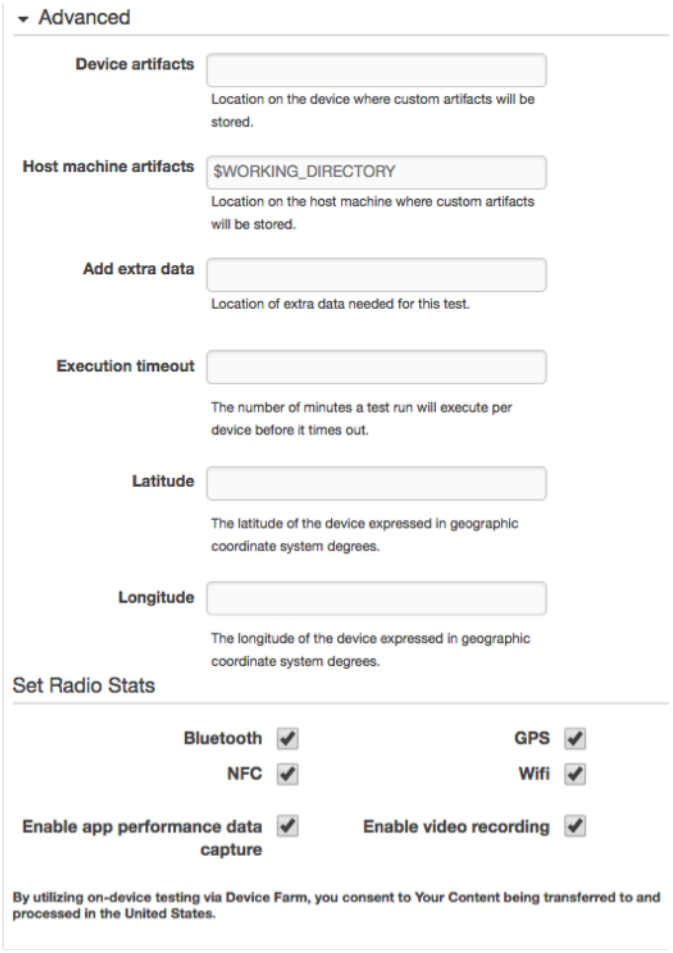



10. 앱 파일 경로에 컴파일된 앱 패키지의 경로를 입력하세요. 경로는 테스트용 입력 아티팩트의 루트와 상대적입니다.
11. 테스트 유형에서 다음 중 하나를 수행하세요.

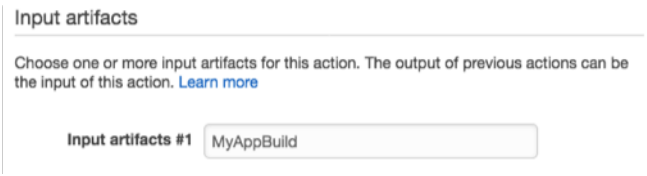
    - 내장된 Device Farm 테스트 중 하나를 사용하는 경우 Device Farm 프로젝트에 구성된 테스트 유형을 선택하세요.
    - Device Farm의 기본 제공 테스트 중 하나를 사용하지 않는 경우 테스트 파일 경로에 테스트 정의 파일의 경로를 입력하세요. 경로는 테스트용 입력 아티팩트의 루트와 상대적입니다.



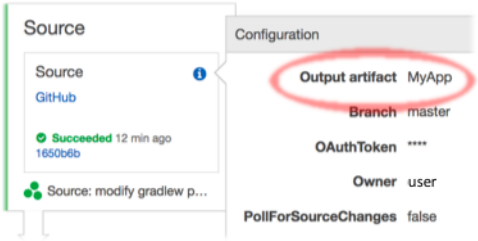

12. 나머지 필드에서 테스트 및 애플리케이션 유형에 적절한 구성을 제공하세요.
13. (선택 사항) 고급에서 테스트 실행을 위한 자세한 구성 정보를 제공하세요.



14. 입력 아티팩트에서 파이프라인의 테스트 스테이지 전에 오는 단계의 출력 아티팩트와 일치하는 입력 아티팩트를 선택하세요.




CodePipeline 콘솔에서 파이프라인 다이어그램의 정보 아이콘 위로 마우스 커서를 가져가면 각 단계의 출력 아티팩트 이름을 알 수 있습니다. 소스 단계에서 파이프라인이 바로 앱을 테스트하면 MyApp을 선택하세요. 파이프라인이 빌드 단계를 포함하면 MyAppBuild를 선택하세요.




15. 패널 하단에서 작업 추가를 선택하세요.
16. CodePipeline 패널 창에서 파이프라인 변경 사항 저장을 선택한 다음 변경 사항 저장을 선택하세요.
17. 변경 사항을 제출하고 파이프라인 빌드를 시작하려면 변경 사항 배포를 선택한 다음 릴리스를 선택하세요.

# AWS Device Farm를 위한 AWS CLI 참조

AWS Command Line Interface(AWS CLI)를 사용하여 Device Farm 명령을 실행하려면 AWS Device Farm AWS CLI 참조를 확인하세요.

AWS CLI에 대한 일반 정보는 AWS Command Line Interface 사용 설명서 및 AWS CLI 명령 참조를 참고하세요.

# AWS Device Farm에 대한 Windows PowerShell 참조

Windows PowerShell을 사용하여 Device Farm 명령을 실행하려면 AWS Tools for Windows PowerShell Cmdlet 참조의 Device Farm Cmdlet 참조를 참조하세요. 자세한 내용은 AWS Tools for Windows PowerShell 사용 설명서의 Windows PowerShell용 AWS 도구 설치를 참조하세요.

# AWS Device Farm 자동화

Device Farm에 프로그래밍 방식 액세스는 실행 예약 또는 실행, 제품군 및 테스트에 대한 아티팩트 다운로드와 같이 수행해야 하는 일반적인 작업을 자동화하는 강력한 방법입니다. AWS SDK 및 AWS CLI는 수행할 수 있는 방법을 제공합니다.

AWS SDK는 Device Farm, Amazon S3 등을 비롯한 모든 AWS 서비스에 액세스할 수 있게 합니다. 자세한 내용을 알아보려면 다음 단원을 참조하세요.

- AWS 도구 및 SDK
- AWS Device Farm API 레퍼런스

## 예제: AWS SDK를 사용하여 Device Farm 실행 시작 및 아티팩트 수집

다음 예제에서는 AWS SDK를 사용하여 Device Farm과 작업하는 방법을 처음부터 끝까지 설명합니다. 이 예제에서는 다음을 수행합니다.

- 테스트 및 애플리케이션 패키지를 Device Farm에 업로드합니다.
- 테스트 실행을 시작하고 완료(또는 실패)될 때까지 기다립니다.
- 테스트 스위트에서 생성한 모든 아티팩트를 다운로드합니다.

이 예제는 타사 `requests` 패키지에 따라 HTTP와 상호 작용합니다.

```
import boto3
import os
import requests
import string
import random
import time
import datetime
import time
import json

# The following script runs a test through Device Farm
#
# Things you have to change:
config = {
```

```
    # This is our app under test.
    "appFilePath":"app-debug.apk",
    "projectArn": "arn:aws:devicefarm:us-
west-2:111122223333:project:1b99bcff-1111-2222-ab2f-8c3c733c55ed",
    # Since we care about the most popular devices, we'll use a curated pool.
    "testSpecArn":"arn:aws:devicefarm:us-west-2::upload:101e31e8-12ac-11e9-ab14-
d663bd873e83",
    "poolArn":"arn:aws:devicefarm:us-west-2::devicepool:082d10e5-d7d7-48a5-ba5c-
b33d66efa1f5",
    "namePrefix":"MyAppTest",
    # This is our test package. This tutorial won't go into how to make these.
    "testPackage":"tests.zip"
}

client = boto3.client('devicefarm')

unique =
 config['namePrefix']+"-"+(datetime.date.today().isoformat())+(''.join(random.sample(string.asc

print(f"The unique identifier for this run is going to be {unique} -- all uploads will
 be prefixed with this.")

def upload_df_file(filename, type_, mime='application/octet-stream'):
    response = client.create_upload(projectArn=config['projectArn'],
        name = (unique)+"_"+os.path.basename(filename),
        type=type_,
        contentType=mime
        )
    # Get the upload ARN, which we'll return later.
    upload_arn = response['upload']['arn']
    # We're going to extract the URL of the upload and use Requests to upload it
    upload_url = response['upload']['url']
    with open(filename, 'rb') as file_stream:
        print(f"Uploading {filename} to Device Farm as {response['upload']['name']}...
 ",end='')
        put_req = requests.put(upload_url, data=file_stream, headers={"content-
type":mime})
        print(' done')
        if not put_req.ok:
            raise Exception("Couldn't upload, requests said we're not ok. Requests
 says: "+put_req.reason)
    started = datetime.datetime.now()
    while True:
```

```
        print(f"Upload of {filename} in state {response['upload']['status']} after
 "+str(datetime.datetime.now() - started))
        if response['upload']['status'] == 'FAILED':
            raise Exception("The upload failed processing. DeviceFarm says reason
 is: \n"+(response['upload']['message'] if 'message' in response['upload'] else
 response['upload']['metadata']))
        if response['upload']['status'] == 'SUCCEEDED':
            break
        time.sleep(5)
        response = client.get_upload(arn=upload_arn)
    print("")
    return upload_arn

our_upload_arn = upload_df_file(config['appFilePath'], "ANDROID_APP")
our_test_package_arn = upload_df_file(config['testPackage'],
 'APPIUM_PYTHON_TEST_PACKAGE')
print(our_upload_arn, our_test_package_arn)
# Now that we have those out of the way, we can start the test run...
response = client.schedule_run(
    projectArn = config["projectArn"],
    appArn = our_upload_arn,
    devicePoolArn = config["poolArn"],
    name=unique,
    test = {
        "type":"APPIUM_PYTHON",
        "testSpecArn": config["testSpecArn"],
        "testPackageArn": our_test_package_arn
        }
    )
run_arn = response['run']['arn']
start_time = datetime.datetime.now()
print(f"Run {unique} is scheduled as arn {run_arn} ")

try:

    while True:
        response = client.get_run(arn=run_arn)
        state = response['run']['status']
        if state == 'COMPLETED' or state == 'ERRORED':
            break
        else:
            print(f" Run {unique} in state {state}, total time
 "+str(datetime.datetime.now()-start_time))
            time.sleep(10)
```

```
except:
    # If something goes wrong in this process, we stop the run and exit.

    client.stop_run(arn=run_arn)
    exit(1)
print(f"Tests finished in state {state} after "+str(datetime.datetime.now() -
 start_time))
# now, we pull all the logs.
jobs_response = client.list_jobs(arn=run_arn)
# Save the output somewhere. We're using the unique value, but you could use something
 else
save_path = os.path.join(os.getcwd(), unique)
os.mkdir(save_path)
# Save the last run information
for job in jobs_response['jobs'] :
    # Make a directory for our information
    job_name = job['name']
    os.makedirs(os.path.join(save_path, job_name), exist_ok=True)
    # Get each suite within the job
    suites = client.list_suites(arn=job['arn'])['suites']
    for suite in suites:
        for test in client.list_tests(arn=suite['arn'])['tests']:
            # Get the artifacts
            for artifact_type in ['FILE','SCREENSHOT','LOG']:
                artifacts = client.list_artifacts(
                    type=artifact_type,
                    arn = test['arn']
                )['artifacts']
                for artifact in artifacts:
                    # We replace : because it has a special meaning in Windows & macos
                    path_to = os.path.join(save_path, job_name, suite['name'],
 test['name'].replace(':','_') )
                    os.makedirs(path_to, exist_ok=True)
                    filename =
 artifact['type']+"_"+artifact['name']+"."+artifact['extension']
                    artifact_save_path = os.path.join(path_to, filename)
                    print("Downloading "+artifact_save_path)
                    with open(artifact_save_path, 'wb') as fn,
 requests.get(artifact['url'],allow_redirects=True) as request:
                        fn.write(request.content)
                    #/for artifact in artifacts
                #/for artifact type in []
            #/ for test in ()[]
        #/ for suite in suites
```

```
    #/ for job in _[]
# done
print("Finished")
```

# Device Farm 오류 문제 해결

이 단원에서는 Device Farm과 관련된 일반적인 문제를 해결하는 데 도움이 되는 오류 메시지와 절차를 찾을 수 있습니다.

주제



## AWS Device Farm의 Android 애플리케이션 테스트 문제 해결

다음 주제에서는 Android 애플리케이션 테스트를 업로드하는 동안 발생하는 오류 메시지를 나열하고 각 오류를 해결하기 위한 방법을 권장합니다.

> **Note**
>
> 다음 지침은 Linux x86_64 및 Mac을 기반으로 합니다.

### ANDROID_APP_UNZIP_FAILED

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
> 애플리케이션을 열 수 없습니다. 파일이 유효한지 확인하고 다시 시도하세요.

응용 프로그램 패키지의 압축을 오류 없이 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 app-debug.apk입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip app-debug.apk
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   Android 애플리케이션 패키지는 다음과 같은 출력이 생성됩니다.

   ```
   .
   |-- AndroidManifest.xml
   |-- classes.dex
   |-- resources.arsc
   |-- assets (directory)
   |-- res (directory)
   `-- META-INF (directory)
   ```

   자세한 내용은 AWS Device Farm에서 Android 테스트 사용 단원을 참조하세요.

## ANDROID_APP_AAPT_DEBUG_BADGING_FAILED

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
> 애플리케이션에 대한 정보를 추출할 수 없습니다. `aapt debug badging` *`<path to your test package>`* 명령을 실행하여 응용 프로그램이 유효한지 확인하고 명령에서 오류가 출력되지 않으면 다시 시도하세요.

업로드 검증 프로세스 중에 AWS Device Farm은 `aapt debug badging` *<path to your package>* 명령 출력에서 정보를 파싱합니다.

Android 애플리케이션에서 이 명령을 성공적으로 실행할 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 app-debug.apk입니다.

- 애플리케이션 패키지를 작업 디렉터리에 복사한 다음 명령을 실행하세요.

```
$ aapt debug badging app-debug.apk
```

Android 애플리케이션 패키지는 다음과 같은 출력이 생성됩니다.

```
package: name='com.amazon.aws.adf.android.referenceapp' versionCode='1'
 versionName='1.0' platformBuildVersionName='5.1.1-1819727'
sdkVersion:'9'
application-label:'ReferenceApp'
application: label='ReferenceApp' icon='res/mipmap-mdpi-v4/ic_launcher.png'
application-debuggable
launchable-activity:
 name='com.amazon.aws.adf.android.referenceapp.Activities.MainActivity'
 label='ReferenceApp' icon=''
uses-feature: name='android.hardware.bluetooth'
uses-implied-feature: name='android.hardware.bluetooth' reason='requested
 android.permission.BLUETOOTH permission, and targetSdkVersion > 4'
main
supports-screens: 'small' 'normal' 'large' 'xlarge'
supports-any-density: 'true'
locales: '--_--'
densities: '160' '213' '240' '320' '480' '640'
```

자세한 내용은 AWS Device Farm에서 Android 테스트 사용 단원을 참조하세요.

## ANDROID_APP_PACKAGE_NAME_VALUE_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 애플리케이션에서 패키지 이름 값을 찾을 수 없습니다. `aapt debug badging` *<path to your test package>* 명령을 실행하여 애플리케이션이 유효한지 확인하고 “package: name” 키워드 뒤에 있는 패키지 이름 값을 찾은 후 다시 시도하세요.

업로드 검증 프로세스 중에 AWS Device Farm은 `aapt debug badging` *<path to your package>* 명령 출력에서 패키지 이름 값을 파싱합니다.

Android 애플리케이션에서 이 명령을 실행하고 패키지 이름 값을 성공적으로 찾을 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 app-debug.apk입니다.

- 작업 디렉터리에 애플리케이션 패키지를 복사한 후 다음 명령을 실행하세요.

  ```
  $ aapt debug badging app-debug.apk | grep "package: name="
  ```

  Android 애플리케이션 패키지는 다음과 같은 출력이 생성됩니다.

  ```
  package: name='com.amazon.aws.adf.android.referenceapp' versionCode='1'
   versionName='1.0' platformBuildVersionName='5.1.1-1819727'
  ```

  자세한 내용은 AWS Device Farm에서 Android 테스트 사용 단원을 참조하세요.

## ANDROID_APP_SDK_VERSION_VALUE_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 애플리케이션에서 SDK 버전 값을 찾을 수 없습니다. `aapt debug badging` *<path to your test package>* 명령을 실행하여 애플리케이션이 유효한지 확인하고 키워드 `sdkVersion` 뒤에 있는 SDK 버전 값을 찾은 후 다시 시도하세요.

업로드 검증 프로세스 중에 AWS Device Farm은 `aapt debug badging` *<path to your package>* 명령 출력에서 SDK 버전 값을 파싱합니다.

Android 애플리케이션에서 이 명령을 실행하고 패키지 이름 값을 성공적으로 찾을 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 app-debug.apk입니다.

- 작업 디렉터리에 애플리케이션 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ aapt debug badging app-debug.apk | grep "sdkVersion"
```

Android 애플리케이션 패키지는 다음과 같은 출력이 생성됩니다.

```
sdkVersion:'9'
```

자세한 내용은 AWS Device Farm에서 Android 테스트 사용 단원을 참조하세요.

## ANDROID_APP_AAPT_DUMP_XMLTREE_FAILED

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 애플리케이션에서 유효한 AndroidManifest.xml을 찾을 수 없습니다. `aapt dump xmltree <path to your test package> AndroidManifest.xml` 명령을 실행하여 테스트 패키지가 유효한지 확인하고 명령에서 오류가 출력되지 않으면 다시 시도하세요.

업로드 검증 프로세스 중에 AWS Device Farm은 `aapt dump xmltree <path to your package> AndroidManifest.xml` 명령을 사용하여 패키지에 포함된 XML 파일의 XML 구문 분석 트리에서 정보를 파싱합니다.

Android 애플리케이션에서 이 명령을 성공적으로 실행할 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 app-debug.apk입니다.

- 작업 디렉터리에 애플리케이션 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ aapt dump xmltree app-debug.apk. AndroidManifest.xml
```

Android 애플리케이션 패키지는 다음과 같은 출력이 생성됩니다.

```
N: android=http://schemas.android.com/apk/res/android
```

```
 E: manifest (line=2)
   A: android:versionCode(0x0101021b)=(type 0x10)0x1
   A: android:versionName(0x0101021c)="1.0" (Raw: "1.0")
   A: package="com.amazon.aws.adf.android.referenceapp" (Raw:
"com.amazon.aws.adf.android.referenceapp")
   A: platformBuildVersionCode=(type 0x10)0x16 (Raw: "22")
   A: platformBuildVersionName="5.1.1-1819727" (Raw: "5.1.1-1819727")
   E: uses-sdk (line=7)
     A: android:minSdkVersion(0x0101020c)=(type 0x10)0x9
     A: android:targetSdkVersion(0x01010270)=(type 0x10)0x16
   E: uses-permission (line=11)
     A: android:name(0x01010003)="android.permission.INTERNET" (Raw:
"android.permission.INTERNET")
   E: uses-permission (line=12)
     A: android:name(0x01010003)="android.permission.CAMERA" (Raw:
"android.permission.CAMERA")
```

자세한 내용은 AWS Device Farm에서 Android 테스트 사용 단원을 참조하세요.

## ANDROID_APP_DEVICE_ADMIN_PERMISSIONS

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 애플리케이션에 디바이스 관리자 권한이 필요한 것으로 확인되었습니다. `aapt dump xmltree <path to your test package> AndroidManifest.xml` 명령을 실행하여 권한이 필요하지 않은지 확인하고 출력에 키워드 `android.permission.BIND_DEVICE_ADMIN`가 포함되어 있지 않은지 확인한 후 다시 시도하세요.

업로드 검증 프로세스 중에 AWS Device Farm은 `aapt dump xmltree <path to your package> AndroidManifest.xml` 명령을 사용하여 패키지 내에 포함된 xml 파일의 xml 구문 분석 트리에서 권한 정보를 파싱합니다.

애플리케이션에 디바이스 관리자 권한이 필요하지 않은지 확인하세요. 다음 예제에서 패키지 이름은 app-debug.apk입니다.

- 작업 디렉터리에 애플리케이션 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ aapt dump xmltree app-debug.apk AndroidManifest.xml
```

그러면 다음과 같은 결과가 표시됩니다.

```
N: android=http://schemas.android.com/apk/res/android
  E: manifest (line=2)
    A: android:versionCode(0x0101021b)=(type 0x10)0x1
    A: android:versionName(0x0101021c)="1.0" (Raw: "1.0")
    A: package="com.amazonaws.devicefarm.android.referenceapp" (Raw:
 "com.amazonaws.devicefarm.android.referenceapp")
    A: platformBuildVersionCode=(type 0x10)0x16 (Raw: "22")
    A: platformBuildVersionName="5.1.1-1819727" (Raw: "5.1.1-1819727")
    E: uses-sdk (line=7)
      A: android:minSdkVersion(0x0101020c)=(type 0x10)0xa
      A: android:targetSdkVersion(0x01010270)=(type 0x10)0x16
    E: uses-permission (line=11)
      A: android:name(0x01010003)="android.permission.INTERNET" (Raw:
 "android.permission.INTERNET")
    E: uses-permission (line=12)
      A: android:name(0x01010003)="android.permission.CAMERA" (Raw:
 "android.permission.CAMERA")
        ……
```

Android 애플리케이션이 유효한 경우 출력에는 다음 `A: android:name(0x01010003)="android.permission.BIND_DEVICE_ADMIN" (Raw: "android.permission.BIND_DEVICE_ADMIN")`이 포함되지 않아야 합니다.

자세한 내용은 AWS Device Farm에서 Android 테스트 사용 단원을 참조하세요.

## Android 애플리케이션의 특정 창에 빈 화면이나 검은색 화면이 표시됩니다.

Android 애플리케이션을 테스트하는 중에 Device Farm의 테스트 동영상 녹화에서 애플리케이션의 특정 창이 검은색 화면으로 표시되는 경우 애플리케이션이 Android FLAG_SECURE 기능을 사용하고 있을 수 있습니다. 이 플래그(Android 공식 문서에 설명되어 있음)는 화면 녹화 도구가 애플리케이션의 특정 창을 녹화하는 것을 방지하는 데 사용됩니다. 따라서 Device Farm의 화면 녹화 기능(자동화 및 원격 액세스 테스트용)에서 이 플래그를 사용하는 경우 애플리케이션 창 대신 검은색 화면이 표시될 수 있습니다.

이 플래그는 개발자가 로그인 페이지와 같은 민감한 정보가 포함된 응용 프로그램 페이지에 자주 사용합니다. 로그인 페이지와 같은 특정 페이지에서 애플리케이션 화면 대신 검은색 화면이 표시되는 경우 개발자에게 문의하여 테스트용으로 이 플래그를 사용하지 않는 애플리케이션 빌드를 구하세요.

또한 Device Farm은 여전히 이 플래그가 있는 응용 프로그램 창과 상호 작용할 수 있습니다. 그러므로 애플리케이션의 로그인 페이지가 검은색 화면으로 표시되는 경우에도 보안 인증 정보를 입력하여 애플리케이션에 로그인할 수 있습니다. 따라서 FLAG_SECURE 플래그로 차단되지 않은 페이지를 볼 수 있습니다.

# AWS Device Farm의 Appium Java JUnit 테스트 문제 해결

다음 항목에서는 Appium Java JUnit 테스트를 업로드하는 동안 발생하는 오류 메시지를 나열하고 각 오류를 해결하기 위한 해결 방법을 권장합니다.

> **Note**
>
> 다음 지침은 Linux x86_64 및 Mac을 기반으로 합니다.

## APPIUM_JAVA_JUNIT_TEST_PACKAGE_PACKAGE_UNZIP_FAILED

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 ZIP 파일을 열 수 없습니다. 파일이 유효한지 확인하고 다시 시도하세요.

오류 없이 테스트 패키지의 압축을 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 zip-with-dependencies.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

유효한 Appium Java JUnit 패키지는 다음과 같은 출력을 생성해야 합니다.

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_JAVA_JUNIT_TEST_PACKAGE_DEPENDENCY_DIR_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 패키지 내에서 dependency-jars 디렉터리를 찾을 수 없습니다. 테스트 패키지의 압축을 풀고 dependency-jars 디렉터리가 패키지 내에 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 zip-with-dependencies.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

Appium Java JUnit 패키지가 유효한 경우 작업 디렉토리 내에서 *dependency-jars* 디렉토리를 찾을 수 있습니다.

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DI

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> dependency-jars 디렉터리 트리에서 JAR 파일을 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음 dependency-jars 디렉터리를 열고 디렉터리에 JAR 파일이 하나 이상 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 zip-with-dependencies.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

Appium Java JUnit 패키지가 유효하다면 *dependency-jars* 디렉토리에서 적어도 하나의 *jar* 파일을 찾을 수 있을 것입니다.

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 패키지에서 *-tests.jar 파일을 찾을 수 없습니다. 테스트 패키지의 압축을 풀고 패키지에 *-tests.jar 파일이 하나 이상 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 zip-with-dependencies.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip zip-with-dependencies.zip
```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

Appium Java JUnit 패키지가 유효하다면 이 예제에서 *acme-android-appium-1.0-SNAPSHOT-tests.jar*와 같은 *jar* 파일을 하나 이상 찾을 수 있을 것입니다. 파일 이름은 다를 수 있지만 *-tests.jar*로 끝나야 합니다.

```
.
|– acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|– acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|– zip-with-dependencies.zip (this .zip file contains all of the items)
`– dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |– com.some-dependency.bar-4.1.jar
      |– com.another-dependency.thing-1.0.jar
      |– joda-time-2.7.jar
      `– log4j-1.2.14.jar
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JA

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 JAR 파일에서 클래스 파일을 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음 테스트 JAR 파일을 압축 해제하고 JAR 파일 내에 클래스 파일이 하나 이상 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 zip-with-dependencies.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip zip-with-dependencies.zip
```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

이 예제에서 *acme-android-appium-1.0-SNAPSHOT-tests.jar*와 같은 jar 파일을 하나 이상 찾을 수 있을 것입니다. 파일 이름은 다를 수 있지만 *-tests.jar*로 끝나야 합니다.

```
.
|– acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|– acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|– zip-with-dependencies.zip (this .zip file contains all of the items)
`– dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |– com.some-dependency.bar-4.1.jar
      |– com.another-dependency.thing-1.0.jar
      |– joda-time-2.7.jar
      `– log4j-1.2.14.jar
```

3. 파일을 성공적으로 추출한 후에는 다음 명령을 실행하여 작업 디렉토리 트리에서 하나 이상의 클래스를 찾아야 합니다.

```
$ tree .
```

다음과 같이 출력되어야 합니다.

```
.
|– acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|– acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
everything built from the ./src/test directory)
|- one-class-file.class
|- folder
|    `-another-class-file.class
|– zip-with-dependencies.zip (this .zip file contains all of the items)
`– dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |– com.some-dependency.bar-4.1.jar
      |– com.another-dependency.thing-1.0.jar
      |– joda-time-2.7.jar
      `– log4j-1.2.14.jar
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_JAVA_JUNIT_TEST_PACKAGE_JUNIT_VERSION_VALUE_UNKNOWN

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> JUnit 버전 값을 찾을 수 없습니다. 테스트 패키지의 압축을 풀고 dependency-jars 디렉터리를 열고 JUnit JAR 파일이 디렉터리 내에 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 zip-with-dependencies.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip zip-with-dependencies.zip
```

2. 패키지의 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉토리 트리 구조를 찾을 수 있습니다.

```
tree .
```

출력은 다음과 같아야 합니다.

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
    |— junit-4.10.jar
    |— com.some-dependency.bar-4.1.jar
    |— com.another-dependency.thing-1.0.jar
    |— joda-time-2.7.jar
    `— log4j-1.2.14.jar
```

Appium Java JUnit 패키지가 유효한 경우 이 예제에서 jar 파일 *junit-4.10.jar*와 유사한 JUnit 종속성 파일을 찾을 수 있습니다. 이름은 *junit* 키워드와 버전 번호(이 예시에서는 4.10)로 구성되어야 합니다.

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> JUnit 버전이 지원하는 최소 버전인 4.10보다 낮은 것으로 확인되었습니다. JUnit 버전을 변경하고 다시 시도하세요.

다음 예제에서 패키지 이름은 zip-with-dependencies.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip zip-with-dependencies.zip
```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

이 예제에서는 *junit-4.10.jar*와 같은 JUnit 종속성 파일을 찾을 수 있고, 이 예제에서의 버전 번호는 4.10입니다.

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— junit-4.10.jar
```

```
|– com.some-dependency.bar-4.1.jar
|– com.another-dependency.thing-1.0.jar
|– joda-time-2.7.jar
`– log4j-1.2.14.jar
```

> **Note**
>
> 테스트 패키지에 지정된 JUnit 버전이 지원되는 최소 버전인 4.10보다 낮으면 테스트가 제대로 실행되지 않을 수 있습니다.

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

# AWS Device Farm의 Appium JUnit 웹 애플리케이션 테스트 문제 해결

다음 항목에서는 Appium Java JUnit 웹 응용 프로그램 테스트를 업로드하는 동안 발생하는 오류 메시지를 나열하고 각 오류를 해결하기 위한 해결 방법을 권장합니다. Device Farm에서 Appium을 사용하는 방법에 대한 자세한 내용은 the section called “Appium” 단원을 참조하세요.

## APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 ZIP 파일을 열 수 없습니다. 파일이 유효한지 확인하고 다시 시도하세요.

오류 없이 테스트 패키지의 압축을 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 zip-with-dependencies.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

유효한 Appium Java JUnit 패키지는 다음과 같은 출력을 생성해야 합니다.

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

## APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_DEPENDENCY_DIR_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 패키지 내에서 dependency-jars 디렉터리를 찾을 수 없습니다. 테스트 패키지의 압축을 풀고 dependency-jars 디렉터리가 패키지 내에 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 zip-with-dependencies.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

Appium Java JUnit 패키지가 유효한 경우 작업 디렉토리 내에서 *dependency-jars* 디렉토리를 찾을 수 있습니다.

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

## APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_IN_DEPENDEN

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> dependency-jars 디렉터리 트리에서 JAR 파일을 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음 dependency-jars 디렉터리를 열고 디렉터리에 JAR 파일이 하나 이상 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 zip-with-dependencies.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

Appium Java JUnit 패키지가 유효하다면 *dependency-jars* 디렉토리에서 적어도 하나의 *jar* 파일을 찾을 수 있을 것입니다.

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

## APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 패키지에서 *-tests.jar 파일을 찾을 수 없습니다. 테스트 패키지의 압축을 풀고 패키지에 *-tests.jar 파일이 하나 이상 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 zip-with-dependencies.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

Appium Java JUnit 패키지가 유효하다면 이 예제에서 *acme-android-appium-1.0-SNAPSHOT-tests.jar*와 같은 *jar* 파일을 하나 이상 찾을 수 있을 것입니다. 파일 이름은 다를 수 있지만 *-tests.jar*로 끝나야 합니다.

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

## APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TES

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 JAR 파일에서 클래스 파일을 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음 테스트 JAR 파일을 압축 해제하고 JAR 파일 내에 클래스 파일이 하나 이상 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 zip-with-dependencies.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

이 예제에서 *acme-android-appium-1.0-SNAPSHOT-tests.jar*와 같은 jar 파일을 하나 이상 찾을 수 있을 것입니다. 파일 이름은 다를 수 있지만 *-tests.jar*로 끝나야 합니다.

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

3. 파일을 성공적으로 추출한 후에는 다음 명령을 실행하여 작업 디렉토리 트리에서 하나 이상의 클래스를 찾아야 합니다.

```
$ tree .
```

다음과 같이 출력되어야 합니다.

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
everything built from the ./src/test directory)
|- one-class-file.class
|- folder
|    `-another-class-file.class
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

# APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_JUNIT_VERSION_VALUE_UNK

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> JUnit 버전 값을 찾을 수 없습니다. 테스트 패키지의 압축을 풀고 dependency-jars 디렉터리를 열고 JUnit JAR 파일이 디렉터리 내에 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 zip-with-dependencies.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. 패키지의 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉토리 트리 구조를 찾을 수 있습니다.

   ```
   tree .
   ```

   출력은 다음과 같아야 합니다.

   ```
   .
   |— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
   |— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
   |— zip-with-dependencies.zip (this .zip file contains all of the items)
   `— dependency-jars  (this is the directory that contains all of your dependencies,
    built as JAR files)
         |— junit-4.10.jar
         |— com.some-dependency.bar-4.1.jar
         |— com.another-dependency.thing-1.0.jar
         |— joda-time-2.7.jar
         `— log4j-1.2.14.jar
   ```

   Appium Java JUnit 패키지가 유효한 경우 이 예제에서 jar 파일 *junit-4.10.jar*와 유사한 JUnit 종속성 파일을 찾을 수 있습니다. 이름은 *junit* 키워드와 버전 번호(이 예시에서는 4.10)로 구성되어야 합니다.

# APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> JUnit 버전이 지원하는 최소 버전인 4.10보다 낮은 것으로 확인되었습니다. JUnit 버전을 변경하고 다시 시도하세요.

다음 예제에서 패키지 이름은 zip-with-dependencies.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   이 예제에서는 *junit-4.10.jar*와 같은 JUnit 종속성 파일을 찾을 수 있고, 이 예제에서는 버전 번호(4.10)를 찾을 수 있습니다.

   ```
   .
   |— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
   |— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
   |— zip-with-dependencies.zip (this .zip file contains all of the items)
   `— dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
         |— junit-4.10.jar
         |— com.some-dependency.bar-4.1.jar
         |— com.another-dependency.thing-1.0.jar
         |— joda-time-2.7.jar
         `— log4j-1.2.14.jar
   ```

> **Note**
>
> 테스트 패키지에 지정된 JUnit 버전이 지원되는 최소 버전인 4.10보다 낮으면 테스트가 제대로 실행되지 않을 수 있습니다.

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

# AWS Device Farm에서의 Appium Java TestNG 테스트 문제 해결

다음 항목에서는 Appium Java TestNG 테스트를 업로드하는 동안 발생하는 오류 메시지를 나열하고 각 오류를 해결하기 위한 해결 방법을 권장합니다.

> **Note**
>
> 다음 지침은 Linux x86_64 및 Mac을 기반으로 합니다.

## APPIUM_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 ZIP 파일을 열 수 없습니다. 파일이 유효한지 확인하고 다시 시도하세요.

오류 없이 테스트 패키지의 압축을 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 zip-with-dependencies.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행합니다.

```
$ unzip zip-with-dependencies.zip
```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

유효한 Appium Java JUnit 패키지는 다음과 같은 출력을 생성해야 합니다.

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 패키지 내에서 `dependency-jars` 디렉터리를 찾을 수 없습니다. 테스트 패키지의 압축을 풀고 `dependency-jars` 디렉터리가 패키지 안에 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 zip-with-dependencies.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행합니다.

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

Appium Java JUnit 패키지가 유효한 경우 작업 디렉토리 내에서 *dependency-jars* 디렉토리를 찾을 수 있습니다.

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> dependency-jars 디렉터리 트리에서 JAR 파일을 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음 dependency-jars 디렉터리를 열고 디렉터리에 JAR 파일이 하나 이상 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 zip-with-dependencies.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행합니다.

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

Appium Java JUnit 패키지가 유효한 경우 *dependency-jars* 디렉토리에서 적어도 하나의 *jar* 파일을 찾을 수 있습니다.

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 패키지에서 *-tests.jar 파일을 찾을 수 없습니다. 테스트 패키지의 압축을 풀고 패키지에 *-tests.jar 파일이 하나 이상 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 zip-with-dependencies.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행합니다.

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

Appium Java JUnit 패키지가 유효하다면 이 예제에서 *acme-android-appium-1.0-SNAPSHOT-tests.jar*와 같은 *jar* 파일을 하나 이상 찾을 수 있을 것입니다. 파일 이름은 다를 수 있지만 *-tests.jar*로 끝나야 합니다.

```
.
|– acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|– acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|– zip-with-dependencies.zip (this .zip file contains all of the items)
`– dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |– com.some-dependency.bar-4.1.jar
      |– com.another-dependency.thing-1.0.jar
      |– joda-time-2.7.jar
      `– log4j-1.2.14.jar
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 JAR 파일에서 클래스 파일을 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음 테스트 JAR 파일을 압축 해제하고 JAR 파일 내에 클래스 파일이 하나 이상 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 zip-with-dependencies.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행합니다.

```
$ unzip zip-with-dependencies.zip
```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

이 예제에서 *acme-android-appium-1.0-SNAPSHOT-tests.jar*와 같은 jar 파일을 하나 이상 찾을 수 있을 것입니다. 파일 이름은 다를 수 있지만 *-tests.jar*로 끝나야 합니다.

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

3. jar 파일에서 파일을 추출하려면 다음 명령을 실행합니다.

```
$ jar xf acme-android-appium-1.0-SNAPSHOT-tests.jar
```

4. 파일을 성공적으로 추출한 후 다음 명령을 실행합니다.

```
$ tree .
```

작업 디렉토리 트리에서 하나 이상의 클래스를 찾아야 합니다.

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
everything built from the ./src/test directory)
|- one-class-file.class
|- folder
|    `— another-class-file.class
|— zip-with-dependencies.zip (this .zip file contains all of the items)
```

```
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

# AWS Device Farm의 Appium Java TestNG 웹 애플리케이션 문제 해결

다음 항목에서는 Appium Java TestNG 웹 응용 프로그램 테스트를 업로드하는 동안 발생하는 오류 메시지를 나열하고 각 오류를 해결하기 위한 해결 방법을 권장합니다.

## APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 ZIP 파일을 열 수 없습니다. 파일이 유효한지 확인하고 다시 시도하세요.

오류 없이 테스트 패키지의 압축을 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 zip-with-dependencies.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   유효한 Appium Java JUnit 패키지는 다음과 같은 출력을 생성해야 합니다.

   ```
   .
   ```

```
|– acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|– acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|– zip-with-dependencies.zip (this .zip file contains all of the items)
`– dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |– com.some-dependency.bar-4.1.jar
      |– com.another-dependency.thing-1.0.jar
      |– joda-time-2.7.jar
      `– log4j-1.2.14.jar
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSIN

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 패키지 내에서 dependency-jars 디렉터리를 찾을 수 없습니다. 테스트 패키지의 압축을 풀고 dependency-jars 디렉터리가 패키지 내에 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 zip-with-dependencies.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   Appium Java JUnit 패키지가 유효한 경우 작업 디렉토리 내에서 *dependency-jars* 디렉토리를 찾을 수 있습니다.

   ```
   .
   ```

```
|– acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|– acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|– zip-with-dependencies.zip (this .zip file contains all of the items)
`– dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |– com.some-dependency.bar-4.1.jar
      |– com.another-dependency.thing-1.0.jar
      |– joda-time-2.7.jar
      `– log4j-1.2.14.jar
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPEND

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> dependency-jars 디렉터리 트리에서 JAR 파일을 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음 dependency-jars 디렉터리를 열고 디렉터리에 JAR 파일이 하나 이상 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 zip-with-dependencies.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   Appium Java JUnit 패키지가 유효한 경우 *dependency-jars* 디렉토리에서 적어도 하나의 *jar* 파일을 찾을 수 있습니다.

   ```
   .
   ```

```
|– acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|– acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|– zip-with-dependencies.zip (this .zip file contains all of the items)
`– dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |– com.some-dependency.bar-4.1.jar
      |– com.another-dependency.thing-1.0.jar
      |– joda-time-2.7.jar
      `– log4j-1.2.14.jar
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 패키지에서 *-tests.jar 파일을 찾을 수 없습니다. 테스트 패키지의 압축을 풀고 패키지에 *-tests.jar 파일이 하나 이상 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 zip-with-dependencies.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   Appium Java JUnit 패키지가 유효하다면 이 예제에서 *acme-android-appium-1.0-SNAPSHOT-tests.jar*와 같은 *jar* 파일을 하나 이상 찾을 수 있을 것입니다. 파일 이름은 다를 수 있지만 *-tests.jar*로 끝나야 합니다.

   ```
   .
   ```

```
|– acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|– acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|– zip-with-dependencies.zip (this .zip file contains all of the items)
`– dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |– com.some-dependency.bar-4.1.jar
      |– com.another-dependency.thing-1.0.jar
      |– joda-time-2.7.jar
      `– log4j-1.2.14.jar
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_T

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 JAR 파일에서 클래스 파일을 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음 테스트 JAR 파일을 압축 해제하고 JAR 파일 내에 클래스 파일이 하나 이상 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 zip-with-dependencies.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   이 예제에서 *acme-android-appium-1.0-SNAPSHOT-tests.jar*와 같은 jar 파일을 하나 이상 찾을 수 있을 것입니다. 파일 이름은 다를 수 있지만 *-tests.jar*로 끝나야 합니다.

   ```
   .
   ```

```
|– acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|– acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|– zip-with-dependencies.zip (this .zip file contains all of the items)
`– dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |– com.some-dependency.bar-4.1.jar
      |– com.another-dependency.thing-1.0.jar
      |– joda-time-2.7.jar
      `– log4j-1.2.14.jar
```

3. jar 파일에서 파일을 추출하려면 다음 명령을 실행하세요.

```
$ jar xf acme-android-appium-1.0-SNAPSHOT-tests.jar
```

4. 파일을 성공적으로 추출한 후 다음 명령을 실행하세요.

```
$ tree .
```

작업 디렉토리 트리에서 하나 이상의 클래스를 찾아야 합니다.

```
.
|– acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|– acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
everything built from the ./src/test directory)
|- one-class-file.class
|- folder
|    `– another-class-file.class
|– zip-with-dependencies.zip (this .zip file contains all of the items)
`– dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |– com.some-dependency.bar-4.1.jar
      |– com.another-dependency.thing-1.0.jar
      |– joda-time-2.7.jar
      `– log4j-1.2.14.jar
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

# AWS Device Farm의 Appium Python 테스트 문제 해결

다음 항목에서는 Appium Python 테스트를 업로드하는 동안 발생하는 오류 메시지를 나열하고 각 오류를 해결하기 위한 해결 방법을 권장합니다.

## APPIUM_PYTHON_TEST_PACKAGE_UNZIP_FAILED

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> Appium 테스트 ZIP 파일을 열 수 없습니다. 파일이 유효한지 확인하고 다시 시도하세요.

오류 없이 테스트 패키지의 압축을 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 test_bundle.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip test_bundle.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   유효한 Appium Python 패키지는 다음과 같은 출력이 생성됩니다.

   ```
   .
   |-- requirements.txt
   |-- test_bundle.zip
   |-- tests (directory)
   |       `-- test_unittest.py
   `-- wheelhouse (directory)
           |-- Appium_Python_Client-0.20-cp27-none-any.whl
           |-- py-1.4.31-py2.py3-none-any.whl
           |-- pytest-2.9.0-py2.py3-none-any.whl
           |-- selenium-2.52.0-cp27-none-any.whl
           `-- wheel-0.26.0-py2.py3-none-any.whl
   ```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> wheelhouse 디렉토리 트리에서 종속성 wheel 파일을 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음 wheelhouse 디렉터리를 열고 디렉터리에 wheel 파일이 하나 이상 있는지 확인한 다음 다시 시도하세요.

오류 없이 테스트 패키지의 압축을 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 test_bundle.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip test_bundle.zip
```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

Appium Python 패키지가 유효하면 *wheelhouse* 디렉토리에서 강조 표시된 파일과 같은 *.whl* 종속 파일을 하나 이상 찾을 수 있습니다.

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|       `-- test_unittest.py
`-- wheelhouse (directory)
        |-- Appium_Python_Client-0.20-cp27-none-any.whl
        |-- py-1.4.31-py2.py3-none-any.whl
        |-- pytest-2.9.0-py2.py3-none-any.whl
        |-- selenium-2.52.0-cp27-none-any.whl
```

```
        `-- wheel-0.26.0-py2.py3-none-any.whl
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_PYTHON_TEST_PACKAGE_INVALID_PLATFORM

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 지원하지 않는 플랫폼으로 지정된 wheel 파일이 하나 이상 발견되었습니다. 테스트 패키지의 압축을 푼 다음 wheelhouse 디렉토리를 열고 wheel 파일 이름이 -any.whl 또는 -linux_x86_64.whl로 끝나는지 확인한 다음 다시 시도하세요.

오류 없이 테스트 패키지의 압축을 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 test_bundle.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip test_bundle.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   Appium Python 패키지가 유효하면 *wheelhouse* 디렉토리에서 강조 표시된 파일과 같은 *.whl* 종속 파일을 하나 이상 찾을 수 있습니다. 파일 이름은 다를 수 있지만 플랫폼을 지정하는 *-any.whl* 또는 *-linux_x86_64.whl*로 끝나야 합니다. windows와 같은 다른 플랫폼은 지원되지 않습니다.

   ```
   .
   |-- requirements.txt
   |-- test_bundle.zip
   |-- tests (directory)
   |       `-- test_unittest.py
   `-- wheelhouse (directory)
           |-- Appium_Python_Client-0.20-cp27-none-any.whl
   ```

```
        |-- py-1.4.31-py2.py3-none-any.whl
        |-- pytest-2.9.0-py2.py3-none-any.whl
        |-- selenium-2.52.0-cp27-none-any.whl
        `-- wheel-0.26.0-py2.py3-none-any.whl
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 패키지에서 테스트 디렉터리를 찾을 수 없습니다. 테스트 패키지의 압축을 풀고 테스트 디렉터리가 패키지 안에 있는지 확인한 다음 다시 시도하세요.

오류 없이 테스트 패키지의 압축을 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 test_bundle.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip test_bundle.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   Appium Python 패키지가 유효하면 작업 디렉토리 내에서 *tests* 디렉토리를 찾을 수 있습니다.

   ```
   .
   |-- requirements.txt
   |-- test_bundle.zip
   |-- tests (directory)
   |       `-- test_unittest.py
   `-- wheelhouse (directory)
           |-- Appium_Python_Client-0.20-cp27-none-any.whl
           |-- py-1.4.31-py2.py3-none-any.whl
           |-- pytest-2.9.0-py2.py3-none-any.whl
   ```

```
        |-- selenium-2.52.0-cp27-none-any.whl
        `-- wheel-0.26.0-py2.py3-none-any.whl
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> tests 디렉터리 트리에서 유효한 테스트 파일을 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음 테스트 디렉터리를 열고 하나 이상의 파일 이름이 “test” 키워드로 시작하거나 끝나는지 확인한 다음 다시 시도하세요.

오류 없이 테스트 패키지의 압축을 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 test_bundle.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip test_bundle.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   Appium Python 패키지가 유효하면 작업 디렉토리 내에서 *tests* 디렉토리를 찾을 수 있습니다. 파일 이름은 다를 수 있지만 *test_#* 시작하거나 *_test.py*로 끝나야 합니다.

   ```
   .
   |-- requirements.txt
   |-- test_bundle.zip
   |-- tests (directory)
   |       `-- test_unittest.py
   `-- wheelhouse (directory)
           |-- Appium_Python_Client-0.20-cp27-none-any.whl
           |-- py-1.4.31-py2.py3-none-any.whl
   ```

```
        |-- pytest-2.9.0-py2.py3-none-any.whl
        |-- selenium-2.52.0-cp27-none-any.whl
        `-- wheel-0.26.0-py2.py3-none-any.whl
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 패키지에서 requirements.txt 파일을 찾을 수 없습니다. 테스트 패키지의 압축을 풀고 requirements.txt 파일이 패키지 안에 들어 있는지 확인한 다음 다시 시도하세요.

오류 없이 테스트 패키지의 압축을 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 test_bundle.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip test_bundle.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   Appium Python 패키지가 유효하면 작업 디렉토리에서 *requirements.txt* 파일을 찾을 수 있습니다.

   ```
   .
   |-- requirements.txt
   |-- test_bundle.zip
   |-- tests (directory)
   |       `-- test_unittest.py
   `-- wheelhouse (directory)
           |-- Appium_Python_Client-0.20-cp27-none-any.whl
           |-- py-1.4.31-py2.py3-none-any.whl
           |-- pytest-2.9.0-py2.py3-none-any.whl
   ```

```
        |-- selenium-2.52.0-cp27-none-any.whl
        `-- wheel-0.26.0-py2.py3-none-any.whl
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> pytest 버전이 지원하는 최소 버전인 2.8.0보다 낮은 것으로 확인되었습니다. requirements.txt 파일에서 pytest 버전을 변경한 후 다시 시도하세요.

오류 없이 테스트 패키지의 압축을 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 test_bundle.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip test_bundle.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   작업 디렉터리에서 *requirements.txt* 파일을 찾을 수 있습니다.

   ```
   .
   |-- requirements.txt
   |-- test_bundle.zip
   |-- tests (directory)
   |       `--test_unittest.py
   `-- wheelhouse (directory)
           |-- Appium_Python_Client-0.20-cp27-none-any.whl
           |-- py-1.4.31-py2.py3-none-any.whl
           |-- pytest-2.9.0-py2.py3-none-any.whl
           |-- selenium-2.52.0-cp27-none-any.whl
   ```

```
        `-- wheel-0.26.0-py2.py3-none-any.whl
```

3. pytest 버전을 가져오려면 다음 명령을 실행하면 됩니다.

```
$ grep "pytest" requirements.txt
```

그러면 다음과 같은 결과가 표시됩니다.

```
pytest==2.9.0
```

여기에는 pytest 버전이 표시되는데, 이 예제에서는 2.9.0입니다. Appium Python 패키지가 유효하다면, 파이테스트 버전은 2.8.0보다 높거나 같아야 합니다.

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS_FAIL

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 종속성 wheel을 설치하지 못했습니다. 테스트 패키지의 압축을 푼 다음 requirements.txt 파일과 wheelhouse 디렉터리를 열고 requirements.txt 파일에 지정된 종속성 wheel이 wheelhouse 디렉터리 내의 종속성 wheel과 정확히 일치하는지 확인한 다음 다시 시도하세요.

패키징 테스트를 위해 Python virtualenv를 설정하는 것이 좋습니다. 다음은 Python virtualenv를 사용하여 가상 환경을 만든 다음 활성화하는 예제 흐름입니다.

```
$ virtualenv workspace
$ cd workspace
$ source bin/activate
```

오류 없이 테스트 패키지의 압축을 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 test_bundle.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip test_bundle.zip
```

2. wheel 파일 설치를 테스트하려면 다음 명령을 실행할 수 있습니다.

```
$ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
requirements.txt
```

유효한 Appium Python 패키지는 다음과 같은 출력이 생성됩니다.

```
Ignoring indexes: https://pypi.python.org/simple
Collecting Appium-Python-Client==0.20 (from -r ./requirements.txt (line 1))
Collecting py==1.4.31 (from -r ./requirements.txt (line 2))
Collecting pytest==2.9.0 (from -r ./requirements.txt (line 3))
Collecting selenium==2.52.0 (from -r ./requirements.txt (line 4))
Collecting wheel==0.26.0 (from -r ./requirements.txt (line 5))
Installing collected packages: selenium, Appium-Python-Client, py, pytest, wheel
  Found existing installation: wheel 0.29.0
    Uninstalling wheel-0.29.0:
      Successfully uninstalled wheel-0.29.0
Successfully installed Appium-Python-Client-0.20 py-1.4.31 pytest-2.9.0
 selenium-2.52.0 wheel-0.26.0
```

3. 가상 환경을 비활성화하려면 다음 명령을 실행하세요.

```
$ deactivate
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> tests 디렉터리에서 테스트를 수집하지 못했습니다. `py.test --collect-only <path to your tests directory>` 명령을 실행하여 테스트 패키지가 유효한지 확인하기 위해 테스트 패키지의 압축을 풀고, 명령에서 오류가 출력되지 않으면 다시 시도하세요.

패키징 테스트를 위해 Python virtualenv를 설정하는 것이 좋습니다. 다음은 Python virtualenv를 사용하여 가상 환경을 만든 다음 활성화하는 예제 흐름입니다.

```
$ virtualenv workspace
$ cd workspace
$ source bin/activate
```

오류 없이 테스트 패키지의 압축을 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 test_bundle.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip test_bundle.zip
   ```

2. wheel 파일을 설치하려면 다음 명령을 실행할 수 있습니다.

   ```
   $ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
   requirements.txt
   ```

3. 테스트를 수집하려면 다음 명령을 실행하면 됩니다.

   ```
   $ py.test --collect-only tests
   ```

   유효한 Appium Python 패키지는 다음과 같은 출력이 생성됩니다.

   ```
   ==================== test session starts ====================
   platform darwin -- Python 2.7.11, pytest-2.9.0, py-1.4.31, pluggy-0.3.1
   rootdir: /Users/zhena/Desktop/Ios/tests, inifile:
   collected 1 items
   <Module 'test_unittest.py'>
     <UnitTestCase 'DeviceFarmAppiumWebTests'>
       <TestCaseFunction 'test_devicefarm'>

   ==================== no tests ran in 0.11 seconds ====================
   ```

4. 가상 환경을 비활성화하려면 다음 명령을 실행하세요.

   ```
   $ deactivate
   ```

   자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

# APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEELS_INSUFFICIENT

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> wheelhouse 디렉토리에서 충분한 wheel 종속성을 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음, wheelhouse 디렉토리를 여세요. requirements.txt 파일에 모든 wheel 종속성이 지정되어 있는지 확인하세요.

오류 없이 테스트 패키지의 압축을 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 test_bundle.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip test_bundle.zip
   ```

2. *requirements.txt* 파일의 길이와 wheelhouse 디렉터리에 있는 *.whl* 종속 파일 수를 확인하세요.

   ```
   $ cat requirements.txt | egrep "." |wc -l
       12
   $ ls wheelhouse/ | egrep ".+\.whl" | wc -l
       11
   ```

   *.whl* 종속 파일 수가 *requirements.txt* 파일의 비어 있지 않은 행 수보다 적은 경우 다음 사항을 확인해야 합니다.

   - *requirements.txt* 파일의 각 행에 해당하는 *.whl* 종속 파일이 있습니다.
   - *requirements.txt* 파일에는 종속성 패키지 이름 이외의 정보가 들어 있는 다른 행이 없습니다.
   - *requirements.txt* 파일에는 종속성 이름이 여러 줄에 중복되지 않으므로 파일의 두 줄이 하나의 *.whl* 종속 파일 하나와 일치할 수 있습니다.

   AWS Device Farm은 *requirements.txt* 파일에서 종속성 패키지에 직접 대응하지 않는 줄(예: `pip install` 명령에 대한 글로벌 옵션을 지정하는 줄)을 지원하지 않습니다. 글로벌 옵션 목록은 요구 사항 파일 형식을 참조하세요.

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

# AWS Device Farm의 Appium Python 웹 애플리케이션 테스트 문제 해결

다음 항목에서는 Appium Python 웹 응용 프로그램 테스트를 업로드하는 동안 발생하는 오류 메시지를 나열하고 각 오류를 해결하기 위한 해결 방법을 권장합니다.

## APPIUM_WEB_PYTHON_TEST_PACKAGE_UNZIP_FAILED

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> Appium 테스트 ZIP 파일을 열 수 없습니다. 파일이 유효한지 확인하고 다시 시도하세요.

오류 없이 테스트 패키지의 압축을 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 test_bundle.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip test_bundle.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   유효한 Appium Python 패키지는 다음과 같은 출력이 생성됩니다.

   ```
   .
   |-- requirements.txt
   |-- test_bundle.zip
   |-- tests (directory)
   |       `-- test_unittest.py
   `-- wheelhouse (directory)
           |-- Appium_Python_Client-0.20-cp27-none-any.whl
   ```

```
        |-- py-1.4.31-py2.py3-none-any.whl
        |-- pytest-2.9.0-py2.py3-none-any.whl
        |-- selenium-2.52.0-cp27-none-any.whl
        `-- wheel-0.26.0-py2.py3-none-any.whl
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_WEB_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> wheelhouse 디렉토리 트리에서 종속성 wheel 파일을 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음 wheelhouse 디렉터리를 열고 디렉터리에 wheel 파일이 하나 이상 있는지 확인한 다음 다시 시도하세요.

오류 없이 테스트 패키지의 압축을 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 test_bundle.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip test_bundle.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   Appium Python 패키지가 유효하면 *wheelhouse* 디렉토리에서 강조 표시된 파일과 같은 *.whl* 종속 파일을 하나 이상 찾을 수 있습니다.

   ```
   .
   |-- requirements.txt
   |-- test_bundle.zip
   |-- tests (directory)
   |       `-- test_unittest.py
   `-- wheelhouse (directory)
   ```

```
        |-- Appium_Python_Client-0.20-cp27-none-any.whl
        |-- py-1.4.31-py2.py3-none-any.whl
        |-- pytest-2.9.0-py2.py3-none-any.whl
        |-- selenium-2.52.0-cp27-none-any.whl
        `-- wheel-0.26.0-py2.py3-none-any.whl
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PLATFORM

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 지원하지 않는 플랫폼으로 지정된 wheel 파일이 하나 이상 발견되었습니다. 테스트 패키지의 압축을 푼 다음 wheelhouse 디렉토리를 열고 wheel 파일 이름이 -any.whl 또는 -linux_x86_64.whl로 끝나는지 확인한 다음 다시 시도하세요.

오류 없이 테스트 패키지의 압축을 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 test_bundle.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip test_bundle.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   Appium Python 패키지가 유효하면 *wheelhouse* 디렉토리에서 강조 표시된 파일과 같은 *.whl* 종속 파일을 하나 이상 찾을 수 있습니다. 파일 이름은 다를 수 있지만 플랫폼을 지정하는 *-any.whl* 또는 *-linux_x86_64.whl*로 끝나야 합니다. windows와 같은 다른 플랫폼은 지원되지 않습니다.

   ```
   .
   |-- requirements.txt
   |-- test_bundle.zip
   ```

```
|-- tests (directory)
|       `-- test_unittest.py
`-- wheelhouse (directory)
        |-- Appium_Python_Client-0.20-cp27-none-any.whl
        |-- py-1.4.31-py2.py3-none-any.whl
        |-- pytest-2.9.0-py2.py3-none-any.whl
        |-- selenium-2.52.0-cp27-none-any.whl
        `-- wheel-0.26.0-py2.py3-none-any.whl
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_WEB_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 패키지에서 tests 디렉터리를 찾을 수 없습니다. 테스트 패키지의 압축을 풀고 테스트 디렉터리가 패키지 안에 있는지 확인한 다음 다시 시도하세요.

오류 없이 테스트 패키지의 압축을 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 test_bundle.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip test_bundle.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   Appium Python 패키지가 유효하면 작업 디렉토리 내에서 *tests* 디렉토리를 찾을 수 있습니다.

   ```
   .
   |-- requirements.txt
   |-- test_bundle.zip
   |-- tests (directory)
   |       `-- test_unittest.py
   ```

```
`-- wheelhouse (directory)
       |-- Appium_Python_Client-0.20-cp27-none-any.whl
       |-- py-1.4.31-py2.py3-none-any.whl
       |-- pytest-2.9.0-py2.py3-none-any.whl
       |-- selenium-2.52.0-cp27-none-any.whl
       `-- wheel-0.26.0-py2.py3-none-any.whl
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> tests 디렉터리 트리에서 유효한 테스트 파일을 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음 테스트 디렉터리를 열고 하나 이상의 파일 이름이 “test” 키워드로 시작하거나 끝나는지 확인한 다음 다시 시도하세요.

오류 없이 테스트 패키지의 압축을 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 test_bundle.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip test_bundle.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   Appium Python 패키지가 유효하면 작업 디렉토리 내에서 *tests* 디렉토리를 찾을 수 있습니다. 파일 이름은 다를 수 있지만 *test_#* 시작하거나 *_test.py*로 끝나야 합니다.

   ```
   .
   |-- requirements.txt
   |-- test_bundle.zip
   |-- tests (directory)
   ```

```
|       `-- test_unittest.py
`-- wheelhouse (directory)
        |-- Appium_Python_Client-0.20-cp27-none-any.whl
        |-- py-1.4.31-py2.py3-none-any.whl
        |-- pytest-2.9.0-py2.py3-none-any.whl
        |-- selenium-2.52.0-cp27-none-any.whl
        `-- wheel-0.26.0-py2.py3-none-any.whl
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_WEB_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISS

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 패키지에서 requirements.txt 파일을 찾을 수 없습니다. 테스트 패키지의 압축을 풀고 requirements.txt 파일이 패키지 안에 들어 있는지 확인한 다음 다시 시도하세요.

오류 없이 테스트 패키지의 압축을 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 test_bundle.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip test_bundle.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   Appium Python 패키지가 유효하면 작업 디렉토리에서 *requirements.txt* 파일을 찾을 수 있습니다.

   ```
   .
   |-- requirements.txt
   |-- test_bundle.zip
   |-- tests (directory)
   ```

```
|       `-- test_unittest.py
`-- wheelhouse (directory)
        |-- Appium_Python_Client-0.20-cp27-none-any.whl
        |-- py-1.4.31-py2.py3-none-any.whl
        |-- pytest-2.9.0-py2.py3-none-any.whl
        |-- selenium-2.52.0-cp27-none-any.whl
        `-- wheel-0.26.0-py2.py3-none-any.whl
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> pytest 버전이 지원하는 최소 버전인 2.8.0보다 낮은 것으로 확인되었습니다. requirements.txt 파일에서 pytest 버전을 변경한 후 다시 시도하세요.

오류 없이 테스트 패키지의 압축을 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 test_bundle.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip test_bundle.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   작업 디렉토리에서 *requirement.txt* 파일을 찾아야 합니다.

   ```
   .
   |-- requirements.txt
   |-- test_bundle.zip
   |-- tests (directory)
   |       `--test_unittest.py
   `-- wheelhouse (directory)
   ```

```
        |-- Appium_Python_Client-0.20-cp27-none-any.whl
        |-- py-1.4.31-py2.py3-none-any.whl
        |-- pytest-2.9.0-py2.py3-none-any.whl
        |-- selenium-2.52.0-cp27-none-any.whl
        `-- wheel-0.26.0-py2.py3-none-any.whl
```

3. Pytest 버전을 가져오려면 다음 명령을 실행할 수 있습니다.

```
$ grep "pytest" requirements.txt
```

그러면 다음과 같은 결과가 표시됩니다.

```
pytest==2.9.0
```

여기에는 pytest 버전이 표시되는데, 이 예제에서는 2.9.0입니다. Appium Python 패키지가 유효하다면, pytest 버전은 2.8.0보다 높거나 같아야 합니다.

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_WEB_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 종속성 wheel을 설치하지 못했습니다. 테스트 패키지의 압축을 푼 다음 requirements.txt 파일과 wheelhouse 디렉터리를 열고 requirements.txt 파일에 지정된 종속성 wheel이 wheelhouse 디렉터리 내의 종속성 wheel과 정확히 일치하는지 확인한 다음 다시 시도하세요.

패키징 테스트를 위해 Python virtualenv를 설정하는 것이 좋습니다. 다음은 Python virtualenv를 사용하여 가상 환경을 만든 다음 활성화하는 예제 흐름입니다.

```
$ virtualenv workspace
$ cd workspace
$ source bin/activate
```

오류 없이 테스트 패키지의 압축을 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 test_bundle.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip test_bundle.zip
```

2. wheel 파일 설치를 테스트하려면 다음 명령을 실행할 수 있습니다.

```
$ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
requirements.txt
```

유효한 Appium Python 패키지는 다음과 같은 출력이 생성됩니다.

```
Ignoring indexes: https://pypi.python.org/simple
Collecting Appium-Python-Client==0.20 (from -r ./requirements.txt (line 1))
Collecting py==1.4.31 (from -r ./requirements.txt (line 2))
Collecting pytest==2.9.0 (from -r ./requirements.txt (line 3))
Collecting selenium==2.52.0 (from -r ./requirements.txt (line 4))
Collecting wheel==0.26.0 (from -r ./requirements.txt (line 5))
Installing collected packages: selenium, Appium-Python-Client, py, pytest, wheel
  Found existing installation: wheel 0.29.0
    Uninstalling wheel-0.29.0:
      Successfully uninstalled wheel-0.29.0
Successfully installed Appium-Python-Client-0.20 py-1.4.31 pytest-2.9.0
 selenium-2.52.0 wheel-0.26.0
```

3. 가상 환경을 비활성화하려면 다음 명령을 실행하세요.

```
$ deactivate
```

자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## APPIUM_WEB_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> tests 디렉터리에서 테스트를 수집하지 못했습니다. “py.test --collect-only <path to your tests directory>” 명령을 실행하여 테스트 패키지가 유효한지 확인하기 위해 테스트 패키지의 압축을 풀고, 명령에서 오류가 출력되지 않으면 다시 시도하세요.

패키징 테스트를 위해 Python virtualenv를 설정하는 것이 좋습니다. 다음은 Python virtualenv를 사용하여 가상 환경을 만든 다음 활성화하는 예제 흐름입니다.

```
$ virtualenv workspace
$ cd workspace
$ source bin/activate
```

오류 없이 테스트 패키지의 압축을 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 test_bundle.zip입니다.

1. 테스트 패키지를 작업 디렉터리에 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip test_bundle.zip
   ```

2. wheel 파일을 설치하려면 다음 명령을 실행할 수 있습니다.

   ```
   $ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
   requirements.txt
   ```

3. 테스트를 수집하려면 다음 명령을 실행할 수 있습니다.

   ```
   $ py.test --collect-only tests
   ```

   유효한 Appium Python 패키지는 다음과 같은 출력이 생성됩니다.

   ```
   ==================== test session starts ====================
   platform darwin -- Python 2.7.11, pytest-2.9.0, py-1.4.31, pluggy-0.3.1
   rootdir: /Users/zhena/Desktop/Ios/tests, inifile:
   collected 1 items
   <Module 'test_unittest.py'>
     <UnitTestCase 'DeviceFarmAppiumWebTests'>
       <TestCaseFunction 'test_devicefarm'>

   ==================== no tests ran in 0.11 seconds ====================
   ```

4. 가상 환경을 비활성화하려면 다음 명령을 실행하세요.

   ```
   $ deactivate
   ```

   자세한 내용은 Appium 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

# AWS Device Farm의 계측 테스트 문제 해결

다음 주제에서는 계측 테스트를 업로드하는 동안 발생하는 오류 메시지를 나열하고 각 오류를 해결하기 위한 해결 방법을 권장합니다.

## INSTRUMENTATION_TEST_PACKAGE_UNZIP_FAILED

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 APK 파일을 열 수 없습니다. 파일이 유효한지 확인하고 다시 시도하세요.

오류 없이 테스트 패키지의 압축을 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 app-debug-androidTest-unaligned.apk입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip app-debug-androidTest-unaligned.apk
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   올바른 계측 테스트 패키지는 다음과 같은 출력이 생성됩니다.

   ```
   .
   |-- AndroidManifest.xml
   |--  classes.dex
   |-- resources.arsc
   |-- LICENSE-junit.txt
   |-- junit (directory)
   `-- META-INF (directory)
   ```

   자세한 내용은 Android 및 AWS Device Farm용 계측 작업 단원을 참조하세요.

# INSTRUMENTATION_TEST_PACKAGE_AAPT_DEBUG_BADGING_FAILED

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 패키지에 대한 정보를 추출할 수 없습니다. “aapt debug badging <path to your test package>” 명령을 실행하여 테스트 패키지가 유효한지 확인하고 명령에서 오류가 출력되지 않으면 다시 시도하세요.

업로드 검증 프로세스 중에 Device Farm은 `aapt debug badging <path to your package>` 명령 출력에서 정보를 구문 분석합니다.

계측 테스트 패키지에서 이 명령을 성공적으로 실행할 수 있는지 확인하세요.

다음 예제에서 패키지 이름은 app-debug-androidTest-unaligned.apk입니다.

- 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

  ```
  $ aapt debug badging app-debug-androidTest-unaligned.apk
  ```

  올바른 계측 테스트 패키지는 다음과 같은 출력이 생성됩니다.

  ```
  package: name='com.amazon.aws.adf.android.referenceapp.test' versionCode=''
   versionName='' platformBuildVersionName='5.1.1-1819727'
  sdkVersion:'9'
  targetSdkVersion:'22'
  application-label:'Test-api'
  application: label='Test-api' icon=''
  application-debuggable
  uses-library:'android.test.runner'
  feature-group: label=''
  uses-feature: name='android.hardware.touchscreen'
  uses-implied-feature: name='android.hardware.touchscreen' reason='default feature
   for all apps'
  supports-screens: 'small' 'normal' 'large' 'xlarge'
  supports-any-density: 'true'
  locales: '--_--'
  densities: '160'
  ```

자세한 내용은 Android 및 AWS Device Farm용 계측 작업 단원을 참조하세요.

## INSTRUMENTATION_TEST_PACKAGE_INSTRUMENTATION_RUNNER_VALU

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> AndroidManifest.xml에서 계측 러너 값을 찾을 수 없습니다. “aapt dump xmltree <path to your test package> AndroidManifest.xml” 명령을 실행하여 테스트 패키지가 유효한지 확인하고 “instrumentation” 키워드 뒤에 있는 계측 러너 값을 찾은 후 다시 시도하세요.

업로드 검증 프로세스 중에 Device Farm은 패키지 내에 포함된 XML 파일의 XML 파스 트리에서 계측 러너 값을 파싱합니다. 다음 `aapt dump xmltree <path to your package> AndroidManifest.xml` 명령을 사용할 수 있습니다.

계측 테스트 패키지에서 이 명령을 실행하여 계측 값을 성공적으로 찾을 수 있는지 확인하세요.

다음 예제에서 패키지 이름은 app-debug-androidTest-unaligned.apk입니다.

- 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ aapt dump xmltree app-debug-androidTest-unaligned.apk AndroidManifest.xml | grep
 -A5 "instrumentation"
```

올바른 계측 테스트 패키지는 다음과 같은 출력이 생성됩니다.

```
E: instrumentation (line=9)
      A: android:label(0x01010001)="Tests for
 com.amazon.aws.adf.android.referenceapp" (Raw: "Tests for
 com.amazon.aws.adf.android.referenceapp")
      A:
 android:name(0x01010003)="android.support.test.runner.AndroidJUnitRunner" (Raw:
 "android.support.test.runner.AndroidJUnitRunner")
      A:
 android:targetPackage(0x01010021)="com.amazon.aws.adf.android.referenceapp" (Raw:
 "com.amazon.aws.adf.android.referenceapp")
      A: android:handleProfiling(0x01010022)=(type 0x12)0x0
```

```
      A: android:functionalTest(0x01010023)=(type 0x12)0x0
```

자세한 내용은 Android 및 AWS Device Farm용 계측 작업 단원을 참조하세요.

## INSTRUMENTATION_TEST_PACKAGE_AAPT_DUMP_XMLTREE_FAILED

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 패키지에서 유효한 AndroidManifest.xml을 찾을 수 없습니다. “aapt dump xmltree <path to your test package>AndroidManifest.xml” 명령을 실행하여 테스트 패키지가 유효한지 확인하고 명령에서 오류가 출력되지 않으면 다시 시도하세요.

업로드 검증 프로세스 중에 Device Farm은 다음 명령 `aapt dump xmltree <path to your package> AndroidManifest.xml`을 사용하여 패키지에 포함된 XML 파일의 XML 분석 트리에서 정보를 구문 분석합니다.

계측 테스트 패키지에서 이 명령을 성공적으로 실행할 수 있는지 확인하세요.

다음 예제에서 패키지 이름은 app-debug-androidTest-unaligned.apk입니다.

- 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

  ```
  $ aapt dump xmltree app-debug-androidTest-unaligned.apk AndroidManifest.xml
  ```

  올바른 계측 테스트 패키지는 다음과 같은 출력이 생성됩니다.

  ```
  N: android=http://schemas.android.com/apk/res/android
    E: manifest (line=2)
      A: package="com.amazon.aws.adf.android.referenceapp.test" (Raw:
   "com.amazon.aws.adf.android.referenceapp.test")
      A: platformBuildVersionCode=(type 0x10)0x16 (Raw: "22")
      A: platformBuildVersionName="5.1.1-1819727" (Raw: "5.1.1-1819727")
      E: uses-sdk (line=5)
        A: android:minSdkVersion(0x0101020c)=(type 0x10)0x9
        A: android:targetSdkVersion(0x01010270)=(type 0x10)0x16
      E: instrumentation (line=9)
  ```

```
     A: android:label(0x01010001)="Tests for
com.amazon.aws.adf.android.referenceapp" (Raw: "Tests for
com.amazon.aws.adf.android.referenceapp")
     A:
android:name(0x01010003)="android.support.test.runner.AndroidJUnitRunner" (Raw:
"android.support.test.runner.AndroidJUnitRunner")
     A:
android:targetPackage(0x01010021)="com.amazon.aws.adf.android.referenceapp" (Raw:
"com.amazon.aws.adf.android.referenceapp")
     A: android:handleProfiling(0x01010022)=(type 0x12)0x0
     A: android:functionalTest(0x01010023)=(type 0x12)0x0
   E: application (line=16)
     A: android:label(0x01010001)=@0x7f020000
     A: android:debuggable(0x0101000f)=(type 0x12)0xffffffff
     E: uses-library (line=17)
       A: android:name(0x01010003)="android.test.runner" (Raw:
"android.test.runner")
```

자세한 내용은 Android 및 AWS Device Farm용 계측 작업 단원을 참조하세요.

## INSTRUMENTATION_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISS

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 패키지에서 패키지 이름을 찾을 수 없습니다. “aapt debug badging <path to your test package>” 명령을 실행하여 테스트 패키지가 유효한지 확인하고 “package: name” 키워드 뒤에 있는 패키지 이름 값을 찾은 후 다시 시도하세요.

업로드 검증 프로세스 중에 Device Farm은 다음 명령 `aapt debug badging <path to your package>`의 출력에서 패키지 이름 값을 구문 분석합니다.

인스트루먼트 테스트 패키지에서 이 명령을 실행하고 패키지 이름 값을 성공적으로 찾을 수 있는지 확인하세요.

다음 예제에서 패키지 이름은 app-debug-androidTest-unaligned.apk입니다.

- 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ aapt debug badging app-debug-androidTest-unaligned.apk | grep "package: name="
```

올바른 계측 테스트 패키지는 다음과 같은 출력이 생성됩니다.

```
package: name='com.amazon.aws.adf.android.referenceapp.test' versionCode=''
 versionName='' platformBuildVersionName='5.1.1-1819727'
```

자세한 내용은 Android 및 AWS Device Farm용 계측 작업 단원을 참조하세요.

# AWS Device Farm의 iOS 애플리케이션 테스트 문제 해결

다음 항목에서는 iOS 애플리케이션 테스트를 업로드하는 동안 발생하는 오류 메시지를 나열하고 각 오류를 해결하기 위한 해결 방법을 권장합니다.

> **Note**
>
> 다음 지침은 Linux x86_64 및 Mac을 기반으로 합니다.

## IOS_APP_UNZIP_FAILED

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 애플리케이션을 열 수 없습니다. 파일이 유효한지 확인하고 다시 시도하세요.

응용 프로그램 패키지의 압축을 오류 없이 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 AWSDeviceFarmiOSReferenceApp.ipa입니다.

1. 작업 디렉터리에 애플리케이션 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. 패키지의 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉토리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

올바른 iOS 애플리케이션 패키지는 다음과 같은 출력이 생성됩니다.

```
.
`-- Payload (directory)
        `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                        |-- Info.plist
                        `-- (any other files)
```

자세한 내용은 AWS Device Farm에서 iOS 테스트 사용 단원을 참조하세요.

## IOS_APP_PAYLOAD_DIR_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 애플리케이션 내에서 Payload 디렉터리를 찾을 수 없습니다. 애플리케이션의 압축을 풀고 Payload 디렉터리가 패키지 안에 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 AWSDeviceFarmiOSReferenceApp.ipa입니다.

1. 작업 디렉터리에 애플리케이션 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. 패키지의 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉토리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   iOS 애플리케이션 패키지가 유효한 경우 작업 디렉토리 내에서 *Payload* 디렉토리를 찾을 수 있습니다.

   ```
   .
   `-- Payload (directory)
   ```

```
        `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                    |-- Info.plist
                    `-- (any other files)
```

자세한 내용은 AWS Device Farm에서 iOS 테스트 사용 단원을 참조하세요.

## IOS_APP_APP_DIR_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> Payload 디렉터리에서 .app 디렉터리를 찾을 수 없습니다. 애플리케이션의 압축을 푼 다음 Payload 디렉터리를 열고 디렉터리 내에 .app 디렉터리가 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 AWSDeviceFarmiOSReferenceApp.ipa입니다.

1. 작업 디렉터리에 애플리케이션 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. 패키지의 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉토리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

iOS 애플리케이션 패키지가 유효한 경우, 이 예제의 *Payload* 디렉터리 내에서 *AWSDeviceFarmiOSReferenceApp.app*과 같은 *.app* 디렉터리를 찾을 수 있습니다.

```
.
`-- Payload (directory)
        `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                    |-- Info.plist
                    `-- (any other files)
```

자세한 내용은 AWS Device Farm에서 iOS 테스트 사용 단원을 참조하세요.

## IOS_APP_PLIST_FILE_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> .app 디렉터리에서 Info.plist 파일을 찾을 수 없습니다. 애플리케이션의 압축을 푼 다음 .app 디렉터리를 열고 Info.plist 파일이 디렉터리 내에 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 AWSDeviceFarmiOSReferenceApp.ipa입니다.

1. 작업 디렉터리에 애플리케이션 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. 패키지의 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉토리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   iOS 애플리케이션 패키지가 유효한 경우 이 예제의 *AWSDeviceFarmiOSReferenceApp.app*과 같은 *.app* 디렉터리에서 *Info.plist* 파일을 찾을 수 있습니다.

   ```
   .
   `-- Payload (directory)
           `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                       |-- Info.plist
                       `-- (any other files)
   ```

   자세한 내용은 AWS Device Farm에서 iOS 테스트 사용 단원을 참조하세요.

## IOS_APP_CPU_ARCHITECTURE_VALUE_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> Info.plist 파일에서 CPU 아키텍처 값을 찾을 수 없습니다. 애플리케이션의 압축을 푼 다음 .app 디렉터리에서 Info.plist 파일을 열고 “UIRequiredDeviceCapacabilities” 키가 지정되었는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 AWSDeviceFarmiOSReferenceApp.ipa입니다.

1. 작업 디렉터리에 애플리케이션 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. 패키지의 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉토리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

   다음 예제의 *AWSDeviceFarmiOSReferenceApp.app*과 같은 *.app* 디렉터리에서 *Info.plist* 파일을 찾을 수 있습니다.

```
.
`-- Payload (directory)
        `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                        |-- Info.plist
                        `-- (any other files)
```

3. CPU 아키텍처 값을 찾으려면 Xcode 또는 Python을 사용하여 Info.plist를 열 수 있습니다.

   Python의 경우 다음 명령을 실행하여 biplist 모듈을 설치할 수 있습니다.

```
$ pip install biplist
```

4. Python을 열고 다음 명령을 실행하세요.

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['UIRequiredDeviceCapabilities']
```

올바른 iOS 애플리케이션 패키지는 다음과 같은 출력이 생성됩니다.

```
['armv7']
```

자세한 내용은 AWS Device Farm에서 iOS 테스트 사용 단원을 참조하세요.

## IOS_APP_PLATFORM_VALUE_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> ⚠ Warning
>
> Info.plist 파일에서 플랫폼 값을 찾을 수 없습니다. 애플리케이션의 압축을 푼 다음 .app 디렉터리에서 Info.plist 파일을 열고 “CFBundleSupportedPlatforms” 키가 지정되었는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 AWSDeviceFarmiOSReferenceApp.ipa입니다.

1. 작업 디렉터리에 애플리케이션 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. 패키지의 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉토리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   다음 예제의 *AWSDeviceFarmiOSReferenceApp.app*과 같은 *.app* 디렉터리에서 *Info.plist* 파일을 찾을 수 있습니다.

   ```
   .
   `-- Payload (directory)
           `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                       |-- Info.plist
                       `-- (any other files)
   ```

3. 플랫폼 값을 찾으려면 Xcode 또는 Python을 사용하여 Info.plist를 열 수 있습니다.

Python의 경우 다음 명령을 실행하여 biplist 모듈을 설치할 수 있습니다.

```
$ pip install biplist
```

4. Python을 열고 다음 명령을 실행하세요.

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['CFBundleSupportedPlatforms']
```

올바른 iOS 애플리케이션 패키지는 다음과 같은 출력이 생성됩니다.

```
['iPhoneOS']
```

자세한 내용은 AWS Device Farm에서 iOS 테스트 사용 단원을 참조하세요.

## IOS_APP_WRONG_PLATFORM_DEVICE_VALUE

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> Info.plist 파일에서 플랫폼 디바이스 값이 잘못된 것을 발견했습니다. 애플리케이션의 압축을 푼 다음 .app 디렉터리에서 Info.plist 파일을 열고 “CFBundleSupportedPlatforms” 키 값에 “simulator”라는 키워드가 포함되어 있지 않은지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 AWSDeviceFarmiOSReferenceApp.ipa입니다.

1. 작업 디렉터리에 애플리케이션 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. 패키지의 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉토리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

다음 예제의 *AWSDeviceFarmiOSReferenceApp.app*과 같은 *.app* 디렉터리에서 *Info.plist* 파일을 찾을 수 있습니다.

```
.
`-- Payload (directory)
        `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                        |-- Info.plist
                        `-- (any other files)
```

3. 플랫폼 값을 찾으려면 Xcode 또는 Python을 사용하여 Info.plist를 열 수 있습니다.

   Python의 경우 다음 명령을 실행하여 biplist 모듈을 설치할 수 있습니다.

```
$ pip install biplist
```

4. Python을 열고 다음 명령을 실행하세요.

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['CFBundleSupportedPlatforms']
```

   올바른 iOS 애플리케이션 패키지는 다음과 같은 출력이 생성됩니다.

```
['iPhoneOS']
```

   iOS 애플리케이션이 유효한 경우 값에 `simulator` 키워드가 포함되지 않아야 합니다.

   자세한 내용은 AWS Device Farm에서 iOS 테스트 사용 단원을 참조하세요.

## IOS_APP_FORM_FACTOR_VALUE_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> Info.plist 파일에서 폼 팩터 값을 찾을 수 없습니다. 응용 프로그램의 압축을 푼 다음 .app 디렉터리에서 Info.plist 파일을 열고 “UIDeviceFamily” 키가 지정되었는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 AWSDeviceFarmiOSReferenceApp.ipa입니다.

1. 작업 디렉터리에 애플리케이션 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. 패키지의 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉토리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

   다음 예제의 *AWSDeviceFarmiOSReferenceApp.app*과 같은 *.app* 디렉터리에서 *Info.plist* 파일을 찾을 수 있습니다.

```
.
`-- Payload (directory)
        `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                        |-- Info.plist
                        `-- (any other files)
```

3. 폼 팩터 값을 찾으려면 Xcode 또는 Python을 사용하여 Info.plist를 열 수 있습니다.

   Python의 경우 다음 명령을 실행하여 biplist 모듈을 설치할 수 있습니다.

```
$ pip install biplist
```

4. Python을 열고 다음 명령을 실행하세요.

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['UIDeviceFamily']
```

올바른 iOS 애플리케이션 패키지는 다음과 같은 출력이 생성됩니다.

```
[1, 2]
```

자세한 내용은 AWS Device Farm에서 iOS 테스트 사용 단원을 참조하세요.

## IOS_APP_PACKAGE_NAME_VALUE_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> Info.plist 파일에서 패키지 이름 값을 찾을 수 없습니다. 애플리케이션의 압축을 푼 다음. app 디렉터리에서 Info.plist 파일을 열고 “CFBundleIdentifier” 키가 지정되었는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 AWSDeviceFarmiOSReferenceApp.ipa입니다.

1. 작업 디렉터리에 애플리케이션 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. 패키지의 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉토리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   다음 예제의 *AWSDeviceFarmiOSReferenceApp.app*과 같은 *.app* 디렉터리에서 *Info.plist* 파일을 찾을 수 있습니다.

   ```
   .
   `-- Payload (directory)
           `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                       |-- Info.plist
                       `-- (any other files)
   ```

3. 패키지 이름 값을 찾으려면 Xcode 또는 Python을 사용하여 Info.plist를 열 수 있습니다.

Python의 경우 다음 명령을 실행하여 biplist 모듈을 설치할 수 있습니다.

```
$ pip install biplist
```

4. Python을 열고 다음 명령을 실행하세요.

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['CFBundleIdentifier']
```

올바른 iOS 애플리케이션 패키지는 다음과 같은 출력이 생성됩니다.

```
Amazon.AWSDeviceFarmiOSReferenceApp
```

자세한 내용은 AWS Device Farm에서 iOS 테스트 사용 단원을 참조하세요.

## IOS_APP_EXECUTABLE_VALUE_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> Info.plist 파일에서 실행 값을 찾을 수 없습니다. 응용 프로그램의 압축을 푼 다음 .app 디렉터리에서 Info.plist 파일을 열고 “CFBundleExecutable” 키가 지정되었는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 AWSDeviceFarmiOSReferenceApp.ipa입니다.

1. 작업 디렉터리에 애플리케이션 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. 패키지의 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉토리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

다음 예제의 *AWSDeviceFarmiOSReferenceApp.app*과 같은 *.app* 디렉터리에서 *Info.plist* 파일을 찾을 수 있습니다.

```
.
`-- Payload (directory)
        `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                        |-- Info.plist
                        `-- (any other files)
```

3. 실행 가능한 값을 찾으려면 Xcode 또는 Python을 사용하여 Info.plist를 열 수 있습니다.

   Python의 경우 다음 명령을 실행하여 biplist 모듈을 설치할 수 있습니다.

```
$ pip install biplist
```

4. Python을 열고 다음 명령을 실행하세요.

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['CFBundleExecutable']
```

   올바른 iOS 애플리케이션 패키지는 다음과 같은 출력이 생성됩니다.

```
AWSDeviceFarmiOSReferenceApp
```

   자세한 내용은 AWS Device Farm에서 iOS 테스트 사용 단원을 참조하세요.

# AWS Device Farm에서의 XCTest 테스트 문제 해결

다음 항목에서는 XCTest 테스트 업로드 중에 발생하는 오류 메시지를 나열하고 각 오류를 해결하기 위한 해결 방법을 권장합니다.

> **Note**
>
> 아래 지침은 MacOS를 사용하고 있다고 가정합니다.

## XCTEST_TEST_PACKAGE_UNZIP_FAILED

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 ZIP 파일을 열 수 없습니다. 파일이 유효한지 확인하고 다시 시도하세요.

응용 프로그램 패키지의 압축을 오류 없이 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 swiftExampleTests.xctest-1.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip swiftExampleTests.xctest-1.zip
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   유효한 XCTest 패키지는 다음과 같은 출력이 생성됩니다.

   ```
   .
   `-- swiftExampleTests.xctest (directory)
                |-- Info.plist
                `-- (any other files)
   ```

   자세한 내용은 iOS용 XCTest 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## XCTEST_TEST_PACKAGE_XCTEST_DIR_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 패키지에서 .xctest 디렉터리를 찾을 수 없습니다. 테스트 패키지의 압축을 풀고, .xctest 디렉터리가 패키지 내에 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 swiftExampleTests.xctest-1.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

XCTest 패키지가 유효하면 작업 디렉토리 내에서 *swiftExampleTests.xctest*와 비슷한 이름의 디렉토리를 찾을 수 있습니다. 이름은 *.xctest*로 끝나야 합니다.

```
.
`-- swiftExampleTests.xctest (directory)
              |-- Info.plist
              `-- (any other files)
```

자세한 내용은 iOS용 XCTest 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## XCTEST_TEST_PACKAGE_PLIST_FILE_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> .xctest 디렉터리에서 Info.plist 파일을 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음 .xctest 디렉터리를 열고 Info.plist 파일이 디렉터리 내에 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 swiftExampleTests.xctest-1.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

XCTest 패키지가 유효한 경우, *.xctest* 디렉토리에서 *Info.plist* 파일을 찾을 수 있습니다. 아래 예시에서는 이 디렉토리를 *swiftExampleTests.xctest*라고 합니다.

```
.
`-- swiftExampleTests.xctest (directory)
              |-- Info.plist
              `-- (any other files)
```

자세한 내용은 iOS용 XCTest 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## XCTEST_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> Info.plist 파일에서 패키지 이름 값을 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음 Info.plist 파일을 열고 “CFBundleIdentifier” 키가 지정되었는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 swiftExampleTests.xctest-1.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

다음 예제의 *swiftExampleTests.xctest*와 같은 *.xctest* 디렉터리에서 *Info.plist* 파일을 찾을 수 있습니다.

```
.
`-- swiftExampleTests.xctest (directory)
              |-- Info.plist
              `-- (any other files)
```

3. 패키지 이름 값을 찾으려면 Xcode 또는 Python을 사용하여 Info.plist를 열 수 있습니다.

   Python의 경우 다음 명령을 실행하여 biplist 모듈을 설치할 수 있습니다.

```
$ pip install biplist
```

4. Python을 열고 다음 명령을 실행하세요.

```
import biplist
info_plist = biplist.readPlist('swiftExampleTests.xctest/Info.plist')
print info_plist['CFBundleIdentifier']
```

   유효한 XCTest 애플리케이션 패키지는 다음과 같은 출력이 생성됩니다.

```
com.amazon.kanapka.swiftExampleTests
```

   자세한 내용은 iOS용 XCTest 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

## XCTEST_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> Info.plist 파일에서 실행 값을 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음 Info.plist 파일을 열고 “CFBundleExecutable” 키가 지정되었는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 swiftExampleTests.xctest-1.zip입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

다음 예제의 *swiftExampleTests.xctest*와 같은 *.xctest* 디렉터리에서 *Info.plist* 파일을 찾을 수 있습니다.

```
.
`-- swiftExampleTests.xctest (directory)
              |-- Info.plist
              `-- (any other files)
```

3. 패키지 이름 값을 찾으려면 Xcode 또는 Python을 사용하여 Info.plist를 열 수 있습니다.

Python의 경우 다음 명령을 실행하여 biplist 모듈을 설치할 수 있습니다.

```
$ pip install biplist
```

4. Python을 열고 다음 명령을 실행하세요.

```
import biplist
info_plist = biplist.readPlist('swiftExampleTests.xctest/Info.plist')
print info_plist['CFBundleExecutable']
```

유효한 XCTest 애플리케이션 패키지는 다음과 같은 출력이 생성됩니다.

```
swiftExampleTests
```

자세한 내용은 iOS용 XCTest 및 AWS Device Farm과 함께 작업 단원을 참조하세요.

# AWS Device Farm의 XCTest UI 테스트 문제 해결

다음 항목에서는 XCTest UI 테스트를 업로드하는 동안 발생하는 오류 메시지를 나열하고 각 오류를 해결하기 위한 해결 방법을 권장합니다.

> **Note**
>
> 다음 지침은 Linux x86_64 및 Mac을 기반으로 합니다.

## XCTEST_UI_TEST_PACKAGE_UNZIP_FAILED

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 IPA 파일을 열 수 없습니다. 파일이 유효한지 확인하고 다시 시도하세요.

응용 프로그램 패키지의 압축을 오류 없이 풀 수 있는지 확인하세요. 다음 예제에서 패키지 이름은 swift-sample-UI.ipa입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   올바른 iOS 애플리케이션 패키지는 다음과 같은 출력이 생성됩니다.

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                       |-- Info.plist
                       |                       `-- (any other files)
                       `-- (any other files)
   ```

   자세한 내용은 XCTest UI 단원을 참조하세요.

# XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> 테스트 패키지 내에서 Payload 디렉터리를 찾을 수 없습니다. 테스트 패키지의 압축을 풀고 Payload 디렉터리가 패키지 안에 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 swift-sample-UI.ipa입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   XCTest UI 패키지가 유효하면 작업 디렉토리 내에서 *Payload* 디렉토리를 찾을 수 있습니다.

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                       |-- Info.plist
                       |                       `-- (any other files)
                       `-- (any other files)
   ```

   자세한 내용은 XCTest UI 단원을 참조하세요.

# XCTEST_UI_TEST_PACKAGE_APP_DIR_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> Payload 디렉터리에서 .app 디렉터리를 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음 Payload 디렉터리를 열고 디렉터리 내에 .app 디렉터리가 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 swift-sample-UI.ipa입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip swift-sample-UI.ipa
```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

XCTest UI 패키지가 유효하면 *Payload* 디렉터리 내의 예제에서 *swift-sampleUITests-Runner.app*과 같은 *.app* 디렉터리를 찾을 수 있습니다.

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                    |-- Info.plist
                    |-- Plugins (directory)
                    |       `swift-sampleUITests.xctest (directory)
                    |                       |-- Info.plist
                    |                       `-- (any other files)
                    `-- (any other files)
```

자세한 내용은 XCTest UI 단원을 참조하세요.

## XCTEST_UI_TEST_PACKAGE_PLUGINS_DIR_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> .app 디렉터리에서 Plugins 디렉터리를 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음 .app 디렉터리를 열고 Plugins 디렉터리가 디렉터리 내에 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 swift-sample-UI.ipa입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip swift-sample-UI.ipa
```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

XCTest UI 패키지가 유효한 경우, *.app* 디렉토리 내에서 *Plugins* 디렉토리를 찾을 수 있습니다. 이 예시에서는 이 디렉토리를 *swift-sampleUITests-Runner.app*이라고 합니다.

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                    |-- Info.plist
                    |-- Plugins (directory)
                    |       `swift-sampleUITests.xctest (directory)
                    |                       |-- Info.plist
                    |                       `-- (any other files)
                    `-- (any other files)
```

자세한 내용은 XCTest UI 단원을 참조하세요.

## XCTEST_UI_TEST_PACKAGE_XCTEST_DIR_MISSING_IN_PLUGINS_DIR

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> plugins 디렉터리에서 .xctest 디렉터리를 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음 plugsin 디렉터리를 열고, .xctest 디렉터리가 디렉터리 내에 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 swift-sample-UI.ipa입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip swift-sample-UI.ipa
```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

XCTest UI 패키지가 유효하면 *Plugins* 디렉터리 안에서 *.xctest* 디렉터리를 찾을 수 있습니다. 이 예제에서는 이 디렉토리를 *swift-sampleUITests.xctest*라고 합니다.

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                    |-- Info.plist
                    |-- Plugins (directory)
                    |       `swift-sampleUITests.xctest (directory)
                    |                       |-- Info.plist
                    |                       `-- (any other files)
                    `-- (any other files)
```

자세한 내용은 XCTest UI 단원을 참조하세요.

## XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> .app 디렉터리에서 Info.plist 파일을 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음 .app 디렉터리를 열고 Info.plist 파일이 디렉터리 내에 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 swift-sample-UI.ipa입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip swift-sample-UI.ipa
```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

XCTest UI 패키지가 유효한 경우, *.app* 디렉토리에서 *Info.plist* 파일을 찾을 수 있습니다. 아래 예시에서는 이 디렉토리를 *swift-sampleUITests-Runner.app*이라고 합니다.

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                        |-- Info.plist
                        |-- Plugins (directory)
                        |       `swift-sampleUITests.xctest (directory)
                        |                         |-- Info.plist
                        |                         `-- (any other files)
                        `-- (any other files)
```

자세한 내용은 XCTest UI 단원을 참조하세요.

## XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING_IN_XCTEST_DIR

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> .xctest 디렉터리에서 Info.plist 파일을 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음 .xctest 디렉터리를 열고 Info.plist 파일이 디렉터리 내에 있는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 swift-sample-UI.ipa입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip swift-sample-UI.ipa
```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

XCTest UI 패키지가 유효한 경우, *.xctest* 디렉토리에서 *Info.plist* 파일을 찾을 수 있습니다. 아래 예시에서는 이 디렉토리를 *swift-sampleUITests.xctest*라고 합니다.

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                    |-- Info.plist
                    |-- Plugins (directory)
                    |       `swift-sampleUITests.xctest (directory)
                    |                       |-- Info.plist
                    |                       `-- (any other files)
                    `-- (any other files)
```

자세한 내용은 XCTest UI 단원을 참조하세요.

## XCTEST_UI_TEST_PACKAGE_CPU_ARCHITECTURE _VALUE_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> Info.plist 파일에서 CPU 아키텍처 값을 계산할 수 없습니다. 테스트 패키지의 압축을 푼 다음 .app 디렉터리에 있는 Info.plist 파일을 열고 “UIRequiredDeviceCapabilities” 키가 지정되었는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 swift-sample-UI.ipa입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip swift-sample-UI.ipa
```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

   다음 예제의 *swift-sampleUITests-Runner.app*과 같은 *.app* 디렉터리에서 *Info.plist* 파일을 찾을 수 있습니다.

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                    |-- Info.plist
                    |-- Plugins (directory)
                    |       `swift-sampleUITests.xctest (directory)
                    |                       |-- Info.plist
                    |                       `-- (any other files)
                    `-- (any other files)
```

3. CPU 아키텍처 값을 찾으려면 Xcode 또는 Python을 사용하여 Info.plist를 열 수 있습니다.

   Python의 경우 다음 명령을 실행하여 biplist 모듈을 설치할 수 있습니다.

```
$ pip install biplist
```

4. Python을 열고 다음 명령을 실행하세요.

```
import biplist
```

```
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/
Info.plist')
print info_plist['UIRequiredDeviceCapabilities']
```

유효한 XCTest UI 패키지는 다음과 같은 출력이 생성됩니다.

```
['armv7']
```

자세한 내용은 XCTest UI 단원을 참조하세요.

## XCTEST_UI_TEST_PACKAGE_PLATFORM_VALUE_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> Info.plist에서 플랫폼 값을 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음 .app 디렉터리에 있는 Info.plist 파일을 열고 “CFBundleSupportedPlatforms” 키가 지정되었는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 swift-sample-UI.ipa입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   다음 예제의 *swift-sampleUITests-Runner.app*과 같은 *.app* 디렉터리에서 *Info.plist* 파일을 찾을 수 있습니다.

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
   ```

```
                        |-- Plugins (directory)
                        |       `swift-sampleUITests.xctest (directory)
                        |                        |-- Info.plist
                        |                        `-- (any other files)
                        `-- (any other files)
```

3. 플랫폼 값을 찾으려면 Xcode 또는 Python을 사용하여 Info.plist를 열 수 있습니다.

   Python의 경우 다음 명령을 실행하여 biplist 모듈을 설치할 수 있습니다.

   ```
   $ pip install biplist
   ```

4. Python을 열고 다음 명령을 실행하세요.

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
   print info_plist['CFBundleSupportedPlatforms']
   ```

   유효한 XCTest UI 패키지는 다음과 같은 출력이 생성됩니다.

   ```
   ['iPhoneOS']
   ```

   자세한 내용은 XCTest UI 단원을 참조하세요.

## XCTEST_UI_TEST_PACKAGE_WRONG_PLATFORM_DEVICE_VALUE

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> Info.plist 파일에서 플랫폼 디바이스 값이 잘못된 것을 발견했습니다. 테스트 패키지의 압축을 푼 다음 .app 디렉터리에 있는 Info.plist 파일을 열고 “CFBundleSupportedPlatforms” 키 값에 “simulator”라는 키워드가 포함되어 있지 않은지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 swift-sample-UI.ipa입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip swift-sample-UI.ipa
```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

다음 예제의 *swift-sampleUITests-Runner.app*과 같은 *.app* 디렉터리에서 *Info.plist* 파일을 찾을 수 있습니다.

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                        |-- Info.plist
                        |-- Plugins (directory)
                        |       `swift-sampleUITests.xctest (directory)
                        |                       |-- Info.plist
                        |                       `-- (any other files)
                        `-- (any other files)
```

3. 플랫폼 값을 찾으려면 Xcode 또는 Python을 사용하여 Info.plist를 열 수 있습니다.

Python의 경우 다음 명령을 실행하여 biplist 모듈을 설치할 수 있습니다.

```
$ pip install biplist
```

4. Python을 열고 다음 명령을 실행하세요.

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['CFBundleSupportedPlatforms']
```

유효한 XCTest UI 패키지는 다음과 같은 출력이 생성됩니다.

```
['iPhoneOS']
```

XCTest UI 패키지가 유효한 경우 값에 `simulator` 키워드가 포함되지 않아야 합니다.

자세한 내용은 XCTest UI 단원을 참조하세요.

# XCTEST_UI_TEST_PACKAGE_FORM_FACTOR_VALUE_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> Info.plist의 폼 팩터 값을 확인할 수 없습니다. 테스트 패키지의 압축을 푼 다음 .app 디렉터리에 있는 Info.plist 파일을 열고 “UIDeviceFamily” 키가 지정되었는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 swift-sample-UI.ipa입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   다음 예제의 *swift-sampleUITests-Runner.app*과 같은 *.app* 디렉터리에서 *Info.plist* 파일을 찾을 수 있습니다.

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                       |-- Info.plist
                       |                       `-- (any other files)
                       `-- (any other files)
   ```

3. 폼 팩터 값을 찾으려면 Xcode 또는 Python을 사용하여 Info.plist를 열 수 있습니다.

   Python의 경우 다음 명령을 실행하여 biplist 모듈을 설치할 수 있습니다.

   ```
   $ pip install biplist
   ```

4. Python을 열고 다음 명령을 실행하세요.

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['UIDeviceFamily']
```

유효한 XCTest UI 패키지는 다음과 같은 출력이 생성됩니다.

```
[1, 2]
```

자세한 내용은 XCTest UI 단원을 참조하세요.

## XCTEST_UI_TEST_PACKAGE_PACKAGE_PACKAGE_NAME_VALUE_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> Info.plist 파일에서 패키지 이름 값을 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음 .app 디렉터리에 있는 Info.plist 파일을 열고 “CFBundleIdentifier” 키가 지정되었는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 swift-sample-UI.ipa입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip swift-sample-UI.ipa
```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

다음 예제의 *swift-sampleUITests-Runner.app*과 같은 *.app* 디렉터리에서 *Info.plist* 파일을 찾을 수 있습니다.

```
.
```

```
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                        |-- Info.plist
                        |-- Plugins (directory)
                        |       `swift-sampleUITests.xctest (directory)
                        |                         |-- Info.plist
                        |                         `-- (any other files)
                        `-- (any other files)
```

3. 패키지 이름 값을 찾으려면 Xcode 또는 Python을 사용하여 Info.plist를 열 수 있습니다.

   Python의 경우 다음 명령을 실행하여 biplist 모듈을 설치할 수 있습니다.

   ```
   $ pip install biplist
   ```

4. Python을 열고 다음 명령을 실행하세요.

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
   print info_plist['CFBundleIdentifier']
   ```

   유효한 XCTest UI 패키지는 다음과 같은 출력이 생성됩니다.

   ```
   com.apple.test.swift-sampleUITests-Runner
   ```

   자세한 내용은 XCTest UI 단원을 참조하세요.

## XCTEST_UI_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> Info.plist 파일에서 실행 값을 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음 .app 디렉터리에 있는 Info.plist 파일을 열고 “CFBundleExecutable” 키가 지정되었는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 swift-sample-UI.ipa입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip swift-sample-UI.ipa
```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

다음 예제의 *swift-sampleUITests-Runner.app*과 같은 *.app* 디렉터리에서 *Info.plist* 파일을 찾을 수 있습니다.

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                        |-- Info.plist
                        |-- Plugins (directory)
                        |       `swift-sampleUITests.xctest (directory)
                        |                         |-- Info.plist
                        |                         `-- (any other files)
                        `-- (any other files)
```

3. 실행 가능한 값을 찾으려면 Xcode 또는 Python을 사용하여 Info.plist를 열 수 있습니다.

Python의 경우 다음 명령을 실행하여 biplist 모듈을 설치할 수 있습니다.

```
$ pip install biplist
```

4. Python을 열고 다음 명령을 실행하세요.

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['CFBundleExecutable']
```

유효한 XCTest UI 패키지는 다음과 같은 출력이 생성됩니다.

```
XCTRunner
```

자세한 내용은 XCTest UI 단원을 참조하세요.

# XCTEST_UI_TEST_PACKAGE_TEST_PACKAGE_TEST_PACKAGE_NAME_VAL

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> .xctest 디렉터리 내의 Info.plist 파일에서 패키지 이름 값을 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음 .xctest 디렉터리에 있는 Info.plist 파일을 열고 “CFBundleIdentifier” 키가 지정되었는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 swift-sample-UI.ipa입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

   ```
   $ tree .
   ```

   다음 예제의 *swift-sampleUITests-Runner.app*과 같은 *.app* 디렉터리에서 *Info.plist* 파일을 찾을 수 있습니다.

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                       |-- Info.plist
                       |                       `-- (any other files)
                       `-- (any other files)
   ```

3. 패키지 이름 값을 찾으려면 Xcode 또는 Python을 사용하여 Info.plist를 열 수 있습니다.

   Python의 경우 다음 명령을 실행하여 biplist 모듈을 설치할 수 있습니다.

   ```
   $ pip install biplist
   ```

4. Python을 열고 다음 명령을 실행하세요.

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Plugins/
swift-sampleUITests.xctest/Info.plist')
print info_plist['CFBundleIdentifier']
```

유효한 XCTest UI 패키지는 다음과 같은 출력이 생성됩니다.

```
com.amazon.swift-sampleUITests
```

자세한 내용은 XCTest UI 단원을 참조하세요.

## XCTEST_UI_TEST_PACKAGE_TEST_EXECUTABLE_VALUE_MISSING

다음 메시지가 표시되면 다음 단계에 따라 문제를 해결하세요.

> **Warning**
>
> .xctest 디렉터리 내의 Info.plist 파일에서 실행 값을 찾을 수 없습니다. 테스트 패키지의 압축을 푼 다음 .xctest 디렉터리에 있는 Info.plist 파일을 열고 “CFBundleExecutable” 키가 지정되었는지 확인한 다음 다시 시도하세요.

다음 예제에서 패키지 이름은 swift-sample-UI.ipa입니다.

1. 작업 디렉터리에 테스트 패키지를 복사한 후 다음 명령을 실행하세요.

```
$ unzip swift-sample-UI.ipa
```

2. 패키지 압축을 성공적으로 푼 후에는 다음 명령을 실행하여 작업 디렉터리 트리 구조를 찾을 수 있습니다.

```
$ tree .
```

다음 예제의 *swift-sampleUITests-Runner.app*과 같은 *.app* 디렉터리에서 *Info.plist* 파일을 찾을 수 있습니다.

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                        |-- Info.plist
                        |-- Plugins (directory)
                        |       `swift-sampleUITests.xctest (directory)
                        |                         |-- Info.plist
                        |                         `-- (any other files)
                        `-- (any other files)
```

3. 실행 가능한 값을 찾으려면 Xcode 또는 Python을 사용하여 Info.plist를 열 수 있습니다.

   Python의 경우 다음 명령을 실행하여 biplist 모듈을 설치할 수 있습니다.

```
$ pip install biplist
```

4. Python을 열고 다음 명령을 실행하세요.

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Plugins/
swift-sampleUITests.xctest/Info.plist')
print info_plist['CFBundleExecutable']
```

   유효한 XCTest UI 패키지는 다음과 같은 출력이 생성됩니다.

```
swift-sampleUITests
```

   자세한 내용은 XCTest UI 단원을 참조하세요.

# AWS Device Farm의 보안

AWS에서 클라우드 보안을 가장 중요하게 생각합니다. AWS 고객은 보안에 매우 민감한 조직의 요구 사항에 부합하도록 구축된 데이터 센터 및 네트워크 아키텍처의 혜택을 누릴 수 있습니다.

보안은 AWS와 귀하의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드의 보안 및 클라우드 내 보안으로 설명합니다.

- 클라우드의 보안: AWS는 AWS 클라우드에서 AWS 서비스를 실행하는 인프라를 보호할 책임이 있습니다. AWS는 안전하게 사용할 수 있는 서비스 또한 제공합니다. 타사 감사자는 AWS 규정 준수 프로그램의 일환으로 보안 효과를 정기적으로 테스트하고 검증합니다. AWS Device Farm에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 규정 준수 프로그램 제공 범위 내 AWS 서비스를 참조하세요.
- 클라우드 내 보안: 귀하의 책임은 귀하가 사용하는 AWS 서비스에 의해 결정됩니다. 또한 귀하는 데이터의 민감도, 회사 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는 Device Farm 사용 시 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 주제에서는 보안 및 규정 준수 목표를 충족하도록 Device Farm을 구성하는 방법을 보여줍니다. 또한 Device Farm 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스를 사용하는 방법을 배우게 됩니다.

주제

# AWS Device Farm의 Identity and Access Management

## 고객

사용 방법 AWS Identity and Access Management (IAM) 은 Device Farm에서 수행하는 작업에 따라 다릅니다.

서비스 사용자: Device Farm 서비스를 사용하여 작업을 수행하는 경우 필요한 보안 인증 정보와 권한을 관리자가 제공합니다. 더 많은 Device Farm 기능을 사용하여 작업을 수행하게 되면 추가 권한이 필요할 수 있습니다. 액세스 권한 관리 방식을 이해하면 적절한 권한을 관리자에게 요청할 수 있습니다. Device Farm의 기능에 액세스할 수 없는 경우 AWS Device Farm 보안 인증 및 액세스 문제 해결 단원을 참조하세요.

서비스 관리자: 회사에서 Device Farm 리소스를 책임지고 있는 경우 Device Farm에 대한 전체 액세스 권한을 가지고 있을 것입니다. 서비스 관리자는 서비스 사용자가 액세스해야 하는 Device Farm 기능과 리소스를 결정합니다. 그런 다음, IAM 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경해야 합니다. 이 페이지의 정보를 검토하여 IAM의 기본 개념을 이해하십시오. 회사가 Device Farm에서 IAM을 사용하는 방법에 대해 자세히 알아보려면 AWS Device Farm이 IAM과 연동되는 방식 단원을 참조하세요.

IAM 관리자: IAM 관리자라면 Device Farm에 대한 액세스 권한 관리 정책 작성 방법을 자세히 알고 싶을 것입니다. IAM에서 사용할 수 있는 CodeDeploy ID 기반 정책 예제를 보려면 AWS Device Farm ID 기반 정책 예제 단원을 참조하세요.

## ID를 통한 인증

인증은 ID 자격 증명을 AWS 사용하여 로그인하는 방법입니다. IAM 사용자로 인증 (로그인 AWS) 하거나 IAM 역할을 맡아 인증 (로그인) 해야 합니다. AWS 계정 루트 사용자

ID 소스를 통해 제공된 자격 증명을 사용하여 페더레이션 ID로 로그인할 수 있습니다. AWS AWS IAM Identity Center (IAM ID 센터) 사용자, 회사의 싱글 사인온 인증, Google 또는 Facebook 자격 증명이 페더레이션 ID의 예입니다. 페더레이션 ID로 로그인할 때 관리자가 이전에 IAM 역할을 사용하여 ID 페더레이션을 설정했습니다. 페더레이션을 사용하여 액세스하는 경우 AWS 간접적으로 역할을 맡게 됩니다.

사용자 유형에 따라 AWS Management Console 또는 AWS 액세스 포털에 로그인할 수 있습니다. 로그인에 대한 자세한 내용은 AWS 로그인 사용 설명서의 내 로그인 방법을 참조하십시오. AWS AWS 계정

AWS 프로그래밍 방식으로 액세스하는 경우 자격 증명을 사용하여 요청에 암호화 방식으로 서명할 수 있는 소프트웨어 개발 키트 (SDK) 와 명령줄 인터페이스 (CLI) 를 AWS 제공합니다. AWS 도구를 사용

하지 않는 경우 요청에 직접 서명해야 합니다. 권장 방법을 사용하여 직접 요청에 서명하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 AWS API 요청 서명을 참조하십시오.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어, AWS 계정의 보안을 강화하기 위해 다단계 인증 (MFA) 을 사용할 것을 권장합니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서의 다중 인증 및 IAM 사용 설명서의 AWS에서 다중 인증(MFA) 사용을 참조하십시오.

### AWS 계정 루트 사용자

계정을 AWS 계정만들 때는 먼저 계정의 모든 AWS 서비스 리소스에 대한 완전한 액세스 권한을 가진 하나의 로그인 ID로 시작합니다. 이 ID를 AWS 계정 루트 사용자라고 하며, 계정을 만들 때 사용한 이메일 주소와 비밀번호로 로그인하여 액세스할 수 있습니다. 일상적인 태스크에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 보안 인증 정보를 보호하고 루트 사용자만 수행할 수 있는 태스크를 수행하는 데 사용하세요. 루트 사용자로 로그인해야 하는 전체 작업 목록은 IAM 사용 설명서의 Tasks that require root user credentials를 참조하십시오.

### IAM 사용자 및 그룹

IAM 사용자는 단일 사용자 또는 애플리케이션에 대한 특정 권한을 가진 사용자 내의 자격 증명입니다. AWS 계정 가능하면 암호 및 액세스 키와 같은 장기 보안 인증이 있는 IAM 사용자를 생성하는 대신 임시 보안 인증을 사용하는 것이 좋습니다. 하지만 IAM 사용자의 장기 보안 인증이 필요한 특정 사용 사례가 있는 경우, 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 장기 보안 인증이 필요한 사용 사례의 경우 정기적으로 액세스 키 교체를 참조하십시오.

IAM 그룹은 IAM 사용자 컬렉션을 지정하는 자격 증명입니다. 사용자는 그룹으로 로그인할 수 없습니다. 그룹을 사용하여 여러 사용자의 권한을 한 번에 지정할 수 있습니다. 그룹을 사용하면 대규모 사용자 집합의 권한을 더 쉽게 관리할 수 있습니다. 예를 들어, IAMAdmins라는 그룹이 있고 이 그룹에 IAM 리소스를 관리할 권한을 부여할 수 있습니다.

사용자는 역할과 다릅니다. 사용자는 한 사람 또는 애플리케이션과 고유하게 연결되지만, 역할은 해당 역할이 필요한 사람이라면 누구나 수임할 수 있습니다. 사용자는 영구적인 장기 보안 인증 정보를 가지고 있지만, 역할은 임시 보안 인증만 제공합니다. 자세한 내용은 IAM 사용 설명서의 IAM 사용자를 만들어야 하는 경우(역할이 아님)를 참조하십시오.

### IAM 역할

IAM 역할은 특정 권한을 가진 사용자 AWS 계정 내의 자격 증명입니다. IAM 사용자와 유사하지만, 특정 개인과 연결되지 않습니다. 역할을 AWS Management Console 전환하여 에서 일시적으로 IAM 역할을 맡을 수 있습니다. AWS CLI 또는 AWS API 작업을 호출하거나 사용자 지정 URL을 사용하여 역

할을 수임할 수 있습니다. 역할 사용 방법에 대한 자세한 내용은 IAM 사용 설명서의 IAM 역할 사용을 참조하십시오.

임시 보안 인증이 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- 페더레이션 사용자 액세스 - 페더레이션 ID에 권한을 부여하려면 역할을 생성하고 해당 역할의 권한을 정의합니다. 페더레이션 ID가 인증되면 역할이 연결되고 역할에 정의된 권한이 부여됩니다. 페더레이션 역할에 대한 자세한 내용은 IAM 사용 설명서의 서드 파티 ID 공급자의 역할 생성 단원을 참조하십시오. IAM Identity Center를 사용하는 경우, 권한 집합을 구성합니다. 인증 후 ID가 액세스할 수 있는 항목을 제어하기 위해 IAM Identity Center는 권한 세트를 IAM의 역할과 연관짓습니다. 권한 세트에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서의 권한 세트를 참조하십시오.
- 임시 IAM 사용자 권한 - IAM 사용자 또는 역할은 IAM 역할을 수임하여 특정 태스크에 대한 다양한 권한을 임시로 받을 수 있습니다.
- 크로스 계정 액세스 - IAM 역할을 사용하여 다른 계정의 사용자(신뢰할 수 있는 보안 주체)가 내 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 계정 간 액세스를 부여하는 기본적인 방법입니다. 그러나 일부 AWS 서비스경우에는 역할을 프록시로 사용하는 대신 정책을 리소스에 직접 연결할 수 있습니다. 크로스 계정 액세스에 대한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 크로스 계정 리소스 액세스를 참조하세요.
- 서비스 간 액세스 — 일부는 다른 AWS 서비스기능을 AWS 서비스 사용합니다. 예를 들어 서비스에서 직접적 호출을 수행하면 일반적으로 해당 서비스는 Amazon EC2에서 애플리케이션을 실행하거나 Amazon S3에 객체를 저장합니다. 서비스는 직접적으로 호출하는 보안 주체의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 태스크를 수행할 수 있습니다.
  - 순방향 액세스 세션 (FAS) — IAM 사용자 또는 역할을 사용하여 작업을 수행하는 경우 보안 AWS 주체로 간주됩니다. 일부 서비스를 사용하는 경우 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는 전화를 거는 주체의 권한을 다운스트림 AWS 서비스서비스에 AWS 서비스 요청하기 위한 요청과 결합하여 사용합니다. FAS 요청은 다른 서비스 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 서비스가 수신한 경우에만 이루어집니다. 이 경우 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.
  - 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 태스크를 수행하기 위해 맡는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 AWS 서비스에 대한 권한을 위임할 역할 생성을 참조하십시오.
  - 서비스 연결 역할 — 서비스 연결 역할은 에 연결된 서비스 역할의 한 유형입니다. AWS 서비스서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은 사용자에게 AWS 계정 표시되며 해당 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.

- Amazon EC2에서 실행되는 애플리케이션 — IAM 역할을 사용하여 EC2 인스턴스에서 실행되고 API 요청을 AWS CLI 하는 애플리케이션의 임시 자격 증명을 관리할 수 있습니다. AWS 이는 EC2 인스턴스 내에 액세스 키를 저장할 때 권장되는 방법입니다. EC2 인스턴스에 AWS 역할을 할당하고 모든 애플리케이션에서 사용할 수 있게 하려면 인스턴스에 연결된 인스턴스 프로필을 생성합니다. 인스턴스 프로파일에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행되는 프로그램이 임시 보안 인증을 얻을 수 있습니다. 자세한 내용은 IAM 사용 설명서의 IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여를 참조하십시오.

IAM 역할을 사용할지 또는 IAM 사용자를 사용할지를 알아보려면 IAM 사용 설명서의 IAM 역할(사용자 대신)을 생성하는 경우를 참조하십시오.

## AWS Device Farm이 IAM과 연동되는 방식

IAM을 사용하여 Device Farm에 대한 액세스를 관리하려면 먼저 어떤 IAM 기능을 Device Farm에 사용할 수 있는지를 이해해야 합니다. Device Farm 및 기타 AWS 서비스가 IAM과 연동되는 방식을 개괄적으로 살펴보려면 IAM 사용 설명서의 IAM과 연동되는AWS 서비스를 참조하십시오.

주제

- Device Farm 자격 증명 기반 정책
- Device Farm 리소스 기반 정책
- 액세스 제어 목록
- Device Farm 태그 기반 권한 부여
- Device Farm IAM 역할

### Device Farm 자격 증명 기반 정책

IAM 자격 증명 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스 및 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. Device Farm은 특정 작업, 리소스 및 조건 키를 지원합니다. JSON 정책에서 사용하는 모든 요소에 대해 알고 싶다면 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

작업

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

JSON 정책의 Action요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 태스크를 설명합니다. 정책 작업은 일반적으로 관련 AWS API 작업과 이름이 같습니다. 일치하는 API 작업이 없는 권한 전용 작업 같은 몇 가지 예외도 있습니다. 정책에서 여러 작업이 필요한 몇 가지 작업도 있습니다. 이러한 추가 작업을 일컬어 종속 작업이라고 합니다.

연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함하십시오.

Device Farm의 정책 작업은 작업 앞에 다음 접두사 devicefarm:를 사용합니다. 예를 들어 Device Farm 데스크톱 브라우저를 통한 CreateTestGridUrl API 작업 테스트로 Selenium 세션을 시작할 수 있는 권한을 부여하려면 정책에 devicefarm:CreateTestGridUrl 작업을 포함합니다. 정책 문에는 Action 또는 NotAction 요소가 포함되어야 합니다. CodeDeploy는 이 서비스로 수행할 수 있는 작업을 설명하는 고유한 작업 세트를 정의합니다.

명령문 하나에 여러 태스크를 지정하려면 다음과 같이 쉼표로 구분합니다.

```
"Action": [
      "devicefarm:action1",
      "devicefarm:action2"
```

와일드카드(*)를 사용하여 여러 작업을 지정할 수 있습니다. 예를 들어, List라는 단어로 시작하는 모든 태스크를 지정하려면 다음 태스크를 포함합니다.

```
"Action": "devicefarm:List*"
```

Device Farm 작업 목록을 보려면 IAM 서비스 인증 참조의 AWS Device Farm에서 정의한 작업을 참조하세요.

리소스

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

Resource JSON 정책 요소는 작업이 적용되는 하나 이상의 개체를 지정합니다. 문장에는 Resource또는 NotResource요소가 반드시 추가되어야 합니다. 모범 사례에 따라 Amazon 리소스 이름(ARN)을 사용하여 리소스를 지정합니다. 리소스 수준 권한이라고 하는 특정 리소스 유형을 지원하는 작업에 대해 이 태스크를 수행할 수 있습니다.

작업 나열과 같이 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"
```

Amazon EC2 인스턴스 리소스에는 다음 ARN이 있습니다.

```
arn:${Partition}:ec2:${Region}:${Account}:instance/${InstanceId}
```

ARN 형식에 대한 자세한 내용은 Amazon 리소스 이름 (ARN) 및 AWS 서비스 네임스페이스를 참조하십시오.

예를 들어 문에서 `i-1234567890abcdef0` 인스턴스를 지정하려면 다음 ARN을 사용합니다.

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:instance/i-1234567890abcdef0"
```

계정에 속하는 모든 인스턴스를 지정하려면 와일드카드(*)를 사용합니다.

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:instance/*"
```

리소스 생성 작업과 같은 일부 Device Farm 작업은 리소스에서 수행할 수 없습니다. 이러한 경우, 와일드카드(*)를 사용해야 합니다.

```
"Resource": "*"
```

다양한 Amazon EC2 API 작업에는 여러 리소스가 관여합니다. 예를 들어 `AttachVolume`은 Amazon EBS 볼륨을 인스턴스에 연결하므로 IAM 사용자에게 볼륨 사용 권한과 인스턴스 사용 권한이 있어야 합니다. 단일 문에서 여러 리소스를 지정하려면 ARN을 쉼표로 구분합니다.

```
"Resource": [
      "resource1",
      "resource2"
```

Device Farm 리소스 유형 및 해당 ARN의 목록을 보려면 IAM 서비스 권한 부여 참조의 AWS Device Farm에서 정의한 리소스 유형을 참조하세요. 각 리소스의 ARN을 지정할 수 있는 작업을 알아보려면 IAM 서비스 권한 부여 참조의 AWS Device Farm가 정의한 작업을 참조하세요.

조건 키

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

Condition 요소(또는 Condition 블록)를 사용하면 정책이 발효되는 조건을 지정할 수 있습니다. Condition 요소는 옵션입니다. 같거나 작음과 같은 조건 연산자를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 생성할 수 있습니다.

한 문에서 여러 Condition 요소를 지정하거나 단일 Condition 요소에서 여러 키를 지정하는 경우, AWS 는 논리적 AND 태스크를 사용하여 평가합니다. 단일 조건 키에 여러 값을 지정하는 경우 는 논리적 OR 연산을 사용하여 조건을 AWS 평가합니다. 명문의 권한을 부여하기 전에 모든 조건을 충족해야 합니다.

조건을 지정할 때 자리 표시자 변수를 사용할 수도 있습니다. 예컨대, IAM 사용자에게 IAM 사용자 이름으로 태그가 지정된 경우에만 리소스에 액세스할 수 있는 권한을 부여할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 IAM 정책 요소: 변수 및 태그를 참조하십시오.

AWS 글로벌 조건 키 및 서비스별 조건 키를 지원합니다. 모든 AWS 글로벌 조건 키를 보려면 IAM 사용 AWS 설명서의 글로벌 조건 컨텍스트 키를 참조하십시오.

Device Farm에서는 자체 조건 키 세트를 정의하고 일부 전역 조건 키 사용도 지원합니다. 모든 AWS 글로벌 조건 키를 보려면 IAM 사용 설명서의AWS 글로벌 조건 컨텍스트 키를 참조하십시오.

Device Farm 조건 키 목록을 보려면 IAM 서비스 권한 부여 참조의 AWS Device Farm을 위한 조건 키를 참조하세요. 조건 키를 사용할 수 있는 작업과 리소스를 알아보려면 IAM 서비스 권한 부여 참조의 AWS Device Farm가 정의한 작업 단원을 참조하세요.

예

CodeDeploy ID 기반 정책 예제를 보려면 AWS Device Farm ID 기반 정책 예제 단원을 참조하세요.

## Device Farm 리소스 기반 정책

Device Farm은 리소스 기반 정책을 지원하지 않습니다.

## 액세스 제어 목록

Device Farm은 액세스 제어 목록(ACL)을 지원하지 않습니다.

## Device Farm 태그 기반 권한 부여

태그를 Device Farm 리소스에 연결하거나 요청을 통해 태그를 Device Farm에 전달할 수 있습니다. 태그에 근거하여 액세스를 제어하려면 aws:ResourceTag/*key-name*, aws:RequestTag/*key-name*

또는 aws:TagKeys 조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다. Device Farm 리소스 태깅에 대한 자세한 내용은 Device Farm에 태그 지정 단원을 참조하세요.

리소스의 태그를 기반으로 리소스에 대한 액세스를 제한하는 자격 증명 기반 정책의 예제는 태그를 기반으로 하는 Device Farm 데스크톱 브라우저 테스트 프로젝트 보기 단원에서 확인할 수 있습니다.

### Device Farm IAM 역할

IAM 역할은 AWS 계정에서 특정 권한을 가진 엔티티입니다.

Device Farm에서 임시 보안 인증 사용

Device Farm에서 임시 보안 인증 사용을 지원합니다.

임시 보안 인증으로 연동하여 로그인하거나, IAM 역할 또는 크로스 계정 역할을 수임할 수 있습니다. AssumeRole또는 GetFederationToken같은 AWS STS API 작업을 호출하여 임시 보안 자격 증명을 얻습니다.

서비스 연결 역할

서비스 연결 역할을 사용하면 AWS 서비스가 다른 서비스의 리소스에 액세스하여 사용자를 대신하여 작업을 완료할 수 있습니다. 서비스 연결 역할은 IAM 계정에 나타나고 서비스가 소유합니다. IAM 관리자는 서비스 연결 역할의 권한을 볼 수는 있지만 편집할 수는 없습니다.

Device Farm은 Device Farm 데스크톱 브라우저 테스트 기능에서 서비스 연결 역할을 사용합니다. 이러한 역할에 대한 자세한 내용은 개발자 안내서의 Device Farm 데스크톱 브라우저 테스트에서 서비스 연결 역할 사용을 참조하십시오.

서비스 역할

Device Farm은 서비스 역할을 지원하지 않습니다.

이 기능을 사용하면 서비스가 사용자를 대신하여 서비스 역할을 수임할 수 있습니다. 이 역할을 사용하면 서비스가 다른 서비스의 리소스에 액세스해 사용자를 대신해 작업을 완료할 수 있습니다. 서비스 역할은 IAM 계정에 나타나고, 해당 계정이 소유합니다. 즉, IAM 관리자가 이 역할에 대한 권한을 변경할 수 있습니다. 그러나 권한을 변경하면 서비스의 기능이 손상될 수 있습니다.

## 정책을 사용한 액세스 관리

정책을 만들고 이를 AWS ID 또는 리소스에 AWS 연결하여 액세스를 제어할 수 있습니다. 정책은 ID 또는 리소스와 연결될 때 AWS 해당 권한을 정의하는 객체입니다. AWS 주도자 (사용자, 루트 사용자 또

는 역할 세션) 가 요청할 때 이러한 정책을 평가합니다. 정책에서 권한은 요청이 허용되거나 거부되는지를 결정합니다. 대부분의 정책은 JSON 문서로 AWS 저장됩니다. JSON 정책 문서의 구조와 콘텐츠에 대한 자세한 내용은 IAM 사용 설명서의 JSON 정책 개요를 참조하십시오.

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

기본적으로, 사용자와 역할에는 어떠한 권한도 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 수임할 수 있습니다.

IAM 정책은 작업을 수행하기 위해 사용하는 방법과 상관없이 작업에 대한 권한을 정의합니다. 예를 들어, `iam:GetRole` 작업을 허용하는 정책이 있다고 가정합니다. 해당 정책을 사용하는 사용자는 AWS Management Console, AWS CLI, 또는 AWS API에서 역할 정보를 가져올 수 있습니다.

## 보안 인증 기반 정책

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자와 역할이 어떤 리소스와 어떤 조건에서 어떤 태스크를 수행할 수 있는지를 제어합니다. ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 IAM 정책 생성을 참조하십시오.

보안 인증 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 사용자, 그룹 또는 역할에 직접 포함됩니다. 관리형 정책은 내 여러 사용자, 그룹 및 역할에 연결할 수 있는 독립형 정책입니다. AWS 계정관리형 정책에는 AWS 관리형 정책과 고객 관리형 정책이 포함됩니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책과 인라인 정책의 선택을 참조하십시오.

다음 표에는 Device Farm AWS 관리형 정책의 개요가 서술되어 있습니다.

| 변경 사항 | 설명 | 날짜 |
| --- | --- | --- |
| AWSDeviceFarmFullAccess | 모든 AWS Device Farm 작업에 대한 전체 액세스 권한을 제공합니다. | 2015년 7월 15일 |
| AWSServiceRoleForDeviceFarmTestGrid | Device Farm이 사용자 대신 AWS 리소스에 액세스할 수 있도록 합니다. | 2021년 5월 20일 |

### 기타 정책 타입

AWS 일반적이지 않은 추가 정책 유형을 지원합니다. 이러한 정책 타입은 더 일반적인 정책 타입에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 – 권한 경계는 자격 증명 기반 정책에 따라 IAM 엔터티(IAM 사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 기능입니다. 개체에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 개체의 보안 인증 기반 정책과 그 권한 경계의 교집합입니다. `Principal` 필드에서 사용자나 역할을 지정하는 리소스 기반 정책은 권한 경계를 통해 제한되지 않습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 권한 경계에 대한 자세한 내용은 IAM 사용 설명서의 IAM 엔터티에 대한 권한 경계를 참조하십시오.
- 서비스 제어 정책 (SCP) - SCP는 조직 또는 조직 단위 (OU) 에 대한 최대 권한을 지정하는 JSON 정책입니다. AWS Organizations AWS Organizations 사업체가 소유한 여러 AWS 계정 개를 그룹화하고 중앙에서 관리하는 서비스입니다. 조직에서 모든 기능을 활성화할 경우, 서비스 제어 정책(SCP)을 임의의 또는 모든 계정에 적용할 수 있습니다. SCP는 각 항목을 포함하여 구성원 계정의 엔티티에 대한 권한을 제한합니다. AWS 계정 루트 사용자조직 및 SCP에 대한 자세한 내용은 AWS Organizations 사용 설명서의 SCP 작동 방식을 참조하십시오.
- 세션 정책 - 세션 정책은 역할 또는 페더레이션 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 결과적으로 얻는 세션의 권한은 사용자 또는 역할의 보안 인증 기반 정책의 교차와 세션 정책입니다. 또한 권한을 리소스 기반 정책에서 가져올 수도 있습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 자세한 내용은 IAM 사용 설명서의 세션 정책을 참조하십시오.

### 여러 정책 타입

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 여러 정책 유형이 관련된 경우 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하십시오.

## AWS Device Farm ID 기반 정책 예제

기본적으로 IAM 사용자 및 역할은 Device Farm 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console AWS CLI, 또는 AWS API를 사용하여 작업을 수행할 수 없습니다. IAM 관리자는 지정된 리소스에서 특정 API 작업을 수행할 수 있는 권한을 사용자와 역할에게 부여하는 IAM 정책을 생성해야 합니다. 그런 다음 관리자는 해당 권한이 필요한 IAM 사용자 또는 그룹에 이러한 정책을 연결해야 합니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 JSON 탭에서 정책 생성을 참조하세요.

주제

- 정책 모범 사례
- 사용자가 자신의 고유한 권한을 볼 수 있도록 허용
- 하나의 Device Farm 데스크톱 브라우저 테스트 프로젝트 액세스
- 태그를 기반으로 하는 Device Farm 데스크톱 브라우저 테스트 프로젝트 보기

## 정책 모범 사례

ID 기반 정책에 따라 계정에서 사용자가 Device Farm 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부가 결정됩니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따릅니다.

- AWS 관리형 정책으로 시작하여 최소 권한 권한으로 이동 — 사용자와 워크로드에 권한을 부여하려면 여러 일반 사용 사례에 권한을 부여하는 AWS 관리형 정책을 사용하세요. 해당 내용은 에서 사용할 수 있습니다. AWS 계정사용 사례에 맞는 AWS 고객 관리형 정책을 정의하여 권한을 더 줄이는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 AWS 관리형 정책 또는 직무에 대한AWS 관리형 정책을 참조하십시오.
- 최소 권한 적용 – IAM 정책을 사용하여 권한을 설정하는 경우, 태스크를 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. IAM을 사용하여 권한을 적용하는 방법에 대한 자세한 정보는 IAM 사용 설명서의 IAM의 정책 및 권한을 참조하십시오.
- IAM 정책의 조건을 사용하여 액세스 추가 제한 – 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어 SSL을 사용하여 모든 요청을 전송해야 한다고 지정하는 정책 조건을 작성할 수 있습니다. 예를 AWS 서비스들어 특정 작업을 통해 서비스 작업을 사용하는 경우 조건을 사용하여 서비스 작업에 대한 액세스 권한을 부여할 수도 AWS CloudFormation있습니다. 자세한 내용은 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하십시오.
- IAM Access Analyzer를 통해 IAM 정책을 확인하여 안전하고 기능적인 권한 보장 - IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 신규 및 기존 정책을 확인합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 추천을 제공하여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 내용은 IAM 사용 설명서의 IAM Access Analyzer 정책 검증을 참조하십시오.
- 멀티 팩터 인증 (MFA) 필요 - IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 위해 AWS 계정 MFA를 활성화하십시오. API 작업을 직접적으로 호출할 때 MFA가 필요하면 정

책에 MFA 조건을 추가합니다. 자세한 내용은 IAM 사용 설명서의 MFA 보호 API 액세스 구성을 참조하십시오.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례를 참조하십시오.

## 사용자가 자신의 고유한 권한을 볼 수 있도록 허용

이 예제는 IAM 사용자가 자신의 사용자 ID에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 API를 사용하여 프로그래밍 방식으로 이 작업을 완료할 수 있는 권한이 포함됩니다. AWS CLI AWS

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
```

```
}
```

## 하나의 Device Farm 데스크톱 브라우저 테스트 프로젝트 액세스

이 예시에서는 AWS 계정의 IAM 사용자에게 Device Farm 데스크톱 브라우저 테스트 프로젝트 중 하나에 대한 액세스 권한을 부여하려고 합니다. `arn:aws:devicefarm:us-west-2:111122223333:testgrid-project:123e4567-e89b-12d3-a456-426655441111` 계정에서 프로젝트와 관련된 항목을 볼 수 있도록 하기 위한 것입니다.

`devicefarm:GetTestGridProject` 엔드포인트 외에도 계정에는 `devicefarm:ListTestGridSessions`, `devicefarm:GetTestGridSession`, `devicefarm:ListTestGridSessionActions` 및 `devicefarm:ListTestGridSessionArtifacts` 엔드포인트가 있어야 합니다.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"GetTestGridProject",
         "Effect":"Allow",
         "Action":[
            "devicefarm:GetTestGridProject"
         ],
         "Resource":"arn:aws:devicefarm:us-west-2:111122223333:testgrid-
project:123e4567-e89b-12d3-a456-426655441111"
      },
      {
         "Sid":"ViewProjectInfo",
         "Effect":"Allow",
         "Action":[
            "devicefarm:ListTestGridSessions",
            "devicefarm:ListTestGridSessionActions",
            "devicefarm:ListTestGridSessionArtifacts"
         ],
         "Resource":"arn:aws:devicefarm:us-west-2:111122223333:testgrid-*:123e4567-
e89b-12d3-a456-426655441111/*"
      }
   ]
}
```

CI 시스템을 사용하는 경우 각 CI 실행기에 고유한 액세스 보안 인증을 제공해야 합니다. 예를 들어 CI 시스템에 `devicefarm:ScheduleRun` 또는 `devicefarm:CreateUpload` 이외의 추가 권한이 필요

할 가능성은 별로 없습니다. 다음 IAM 정책은 CI 실행기가 업로드를 생성하고 해당 업로드를 사용하여 테스트 실행을 예약함으로써 새로운 Device Farm 기본 앱 테스트를 시작할 수 있도록 허용하는 최소 정책을 보여줍니다.

```
{
   "Version":"2012-10-17",
   "Statement": [
      {
         "$id":"scheduleTestRuns",
         "effect":"Allow",
         "Action": [ "devicefarm:CreateUpload","devicefarm:ScheduleRun" ],
         "Resource": [
            "arn:aws:devicefarm:us-west-2:111122223333:project:123e4567-e89b-12d3-
a456-426655440000",
            "arn:aws:devicefarm:us-west-2:111122223333:*:123e4567-e89b-12d3-
a456-426655440000/*",
            ]
         }
    ]
}
```

## 태그를 기반으로 하는 Device Farm 데스크톱 브라우저 테스트 프로젝트 보기

ID 기반 정책의 조건을 사용하여 태그를 기반으로 Device Farm 리소스에 대한 액세스를 제어할 수 있습니다. 이 예제에서는 프로젝트 및 세션 보기를 허용하는 정책을 생성하는 방법을 보여줍니다. 요청된 리소스의 `Owner` 태그가 요청 계정의 사용자 이름과 일치하면 권한이 부여됩니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ListTestGridProjectSessions",
            "Effect": "Allow",
            "Action": [
              "devicefarm:ListTestGridSession*",
              "devicefarm:GetTestGridSession",
              "devicefarm:ListTestGridProjects"
              ],
            "Resource": [
               "arn:aws:devicefarm:us-west-2:testgrid-project:*/*"
```

```
                "arn:aws:devicefarm:us-west-2:testgrid-session:*/*"
                ],
            "Condition": {
                "StringEquals": {"aws:TagKey/Owner":"${aws:username}"}
            }
        }
    ]
}
```

이 정책을 계정의 IAM 사용자에게 연결할 수 있습니다. 이름이 `richard-roe`인 사용자가 Device Farm 프로젝트 또는 세션을 보려고 하면 프로젝트에 `Owner=richard-roe` 또는 `owner=richard-roe` 태그를 지정해야 합니다. 그렇지 않으면 사용자는 액세스가 거부됩니다. 조건 키 이름은 대소문자를 구분하지 않기 때문에 조건 태그 키 `Owner`는 `Owner` 및 `owner` 모두와 일치합니다. 자세한 정보는 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.

## AWS Device Farm 보안 인증 및 액세스 문제 해결

다음 정보를 사용하여 Device Farm 및 IAM에서 발생할 수 있는 공통적인 문제를 진단하고 수정할 수 있습니다.

### Device Farm에서 작업을 수행할 권한이 없음

작업을 수행할 권한이 AWS Management Console 없다는 오류 메시지가 표시되는 경우 관리자에게 도움을 요청해야 합니다. 관리자는 사용자 이름과 암호를 제공한 사람입니다.

다음 예제 오류는 IAM 사용자인 `mateojackson`가 콘솔을 사용하여 실행에 대한 세부 정보를 보려고 하지만 `devicefarm:GetRun` 권한이 없는 경우에 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 devicefarm:GetRun on resource: arn:aws:devicefarm:us-west-2:123456789101:run:123e4567-
e89b-12d3-a456-426655440000/123e4567-e89b-12d3-a456-426655441111
```

이 경우 Mateo는 `devicefarm:GetRun` 작업을 사용하여 `arn:aws:devicefarm:us-west-2:123456789101:run:123e4567-e89b-12d3-a456-426655440000/123e4567-e89b-12d3-a456-426655441111` 리소스에서 `devicefarm:GetRun`에 액세스할 수 있도록 정책을 업데이트할 것을 관리자에게 요청합니다.

### 저는 IAM을 수행할 권한이 없습니다. PassRole

`iam:PassRole` 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 Device Farm에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

새 서비스 역할 또는 서비스 연결 역할을 만드는 대신 기존 역할을 해당 서비스에 전달할 AWS 서비스 수 있는 기능도 있습니다. 이렇게 하려면 사용자가 서비스에 역할을 전달할 수 있는 권한을 가지고 있어야 합니다.

다음 예제 오류는 marymajor(이)라는 IAM 사용자가 콘솔을 사용하여 Device Farm에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 있어야 합니다. Mary는 서비스에 역할을 전달할 수 있는 권한을 가지고 있지 않습니다.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

이 경우, Mary가 iam:PassRole 작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요하면 관리자에게 문의하세요. AWS 관리자는 로그인 보안 인증 정보를 제공한 사람입니다.

## 액세스 키를 보아야 합니다.

IAM 사용자 액세스 키를 생성한 후에는 언제든지 액세스 키 ID를 볼 수 있습니다. 하지만 보안 액세스 키는 다시 볼 수 없습니다. 보안 액세스 키를 잃어버린 경우 새로운 액세스 키 페어를 생성해야 합니다.

액세스 키는 액세스 키 ID(예: AKIAIOSFODNN7EXAMPLE)와 보안 액세스 키(예: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY)의 두 가지 부분으로 구성됩니다. 사용자 이름 및 암호와 같이 액세스 키 ID와 보안 액세스 키를 함께 사용하여 요청을 인증해야 합니다. 사용자 이름과 암호를 관리하는 것처럼 안전하게 액세스 키를 관리합니다.

> **Important**
>
> 정식 사용자 ID를 찾는 데 도움이 되더라도 액세스 키를 타사에 제공하지 마시기 바랍니다. 이렇게 하면 다른 사람에게 내 계정에 대한 영구 액세스 권한을 부여할 수 있습니다 AWS 계정.

액세스 키 페어를 생성할 때는 액세스 키 ID와 보안 액세스 키를 안전한 위치에 저장하라는 메시지가 나타납니다. 보안 액세스 키는 생성할 때만 사용할 수 있습니다. 하지만 보안 액세스 키를 잃어버린 경우 새로운 액세스 키를 IAM 사용자에게 추가해야 합니다. 최대 두 개의 액세스 키를 가질 수 있습니다. 이미 두 개가 있는 경우 새로 생성하려면 먼저 키 페어 하나를 삭제해야 합니다. 지침을 보려면 IAM 사용 설명서의 액세스 키 관리를 참조하십시오.

### 관리자인데, 다른 사용자가 Device Farm에 액세스할 수 있게 허용하려고 합니다

다른 사용자가 Device Farm에 액세스하도록 허용하려면 액세스 권한이 필요한 사용자나 애플리케이션에 대한 IAM 엔터티(사용자 또는 역할)를 생성해야 합니다. 다른 사용자들은 해당 엔터티에 대한 보안 인증을 사용해 AWS에 액세스합니다. 그런 다음 Device Farm에 대한 올바른 권한을 부여하는 정책을 엔터티에 연결해야 합니다.

바로 시작하려면 IAM 사용 설명서의 첫 번째 IAM 위임 사용자 및 그룹 생성을 참조하십시오.

### 내 AWS 계정 외부의 사람이 내 리소스에 액세스하도록 허용하려고 합니다.

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제어 목록(ACL)을 지원하는 서비스의 경우 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세히 알아보려면 다음을 참조하십시오.

- Device Farm에서 이러한 기능을 지원하는지 여부를 알아보려면 AWS Device Farm이 IAM과 연동되는 방식 단원을 참조하세요.
- 소유한 리소스에 대한 액세스 권한을 AWS 계정 부여하는 방법을 알아보려면 IAM 사용 설명서의 다른 AWS 계정 IAM 사용자에게 액세스 권한 제공을 참조하십시오.
- 제3자에게 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 타사 AWS 계정AWS 계정 소유에 대한 액세스 제공을 참조하십시오.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 외부에서 인증된 사용자에게 액세스 권한 제공(자격 증명 페더레이션)을 참조하십시오.
- 크로스 계정 액세스에 대한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 크로스 계정 리소스 액세스를 참조하세요.

## AWS Device Farm의 규정 준수 확인

타사 감사자는 여러 AWS 규정 준수 프로그램의 일환으로 AWS Device Farm의 보안 및 규정 준수를 평가합니다. 여기에는 SOC, PCI, FedRAMP, HIPAA 등이 포함됩니다. AWS Device Farm은 AWS 규정 준수 프로그램의 범위 내에 있지 않습니다.

특정 규정 준수 프로그램의 범위 내에 있는 AWS 서비스 목록은 규정 준수 프로그램 제공 범위 내 AWS 서비스를 참조하세요. 일반적인 정보는 AWS 규정 준수 프로그램을 참조하세요.

AWS Artifact를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다. 자세한 내용은 AWS Artifact에서 보고서 다운로드를 참조하세요.

Device Farm 사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표 및 관련 법률과 규정에 따라 결정됩니다. AWS에서는 규정 준수를 지원할 다음과 같은 리소스를 제공합니다.

- 보안 및 규정 준수 빠른 시작 안내서: 이 배포 안내서에서는 아키텍처 고려 사항에 관해 설명하고 AWS에서 보안 및 규정 준수에 중점을 둔 기본 환경을 배포하기 위한 단계를 제공합니다.
- AWS 규정 준수 리소스: 고객 조직이 속한 산업 및 위치에 적용될 수 있는 워크북 및 가이드 콜렉션입니다.
- AWS Config 개발자 가이드의 규칙을 사용하여 리소스 평가: AWS Config를 사용하여 리소스 구성이 내부 사례, 업계 지침, 규정을 얼마나 잘 준수하는지 평가합니다.
- AWS Security Hub: 이 AWS 서비스는 보안 산업 표준 및 모범 사례 규정 준수 여부를 확인하는 데 도움이 되도록 AWS 내 보안 상태를 종합적으로 보여줍니다.

# AWS Device Farm의 데이터 보호

AWS 공동 책임 모델은 AWS Device Farm(Device Farm)의 데이터 보호에 적용됩니다. 이 모델에서 설명하는 것처럼 AWS는 모든 AWS 클라우드를 실행하는 글로벌 인프라를 보호할 책임이 있습니다. 이 인프라에서 호스팅되는 콘텐츠에 대한 제어를 유지하는 것은 사용자의 책임입니다. 사용하는 AWS 서비스의 보안 구성과 관리 작업에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 대한 자세한 내용은 데이터 프라이버시 FAQ를 참조하십시오. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 AWS 공동 책임 모델 및 GDPR 블로그 게시물을 참조하세요.

데이터를 보호하려면 AWS 계정 보안 인증 정보를 보호하고 AWS IAM Identity Center 또는 AWS Identity and Access Management(IAM)을 통해 개별 사용자 계정을 설정하는 것이 좋습니다. 이 방식을 사용하면 각 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 다중 인증(MFA)을 사용합니다.
- SSL/TLS를 사용하여 AWS리소스와 통신합니다. TLS 1.2가 필수이며 TLS 1.3을 권장합니다.
- AWS CloudTrail(으)로 API 및 사용자 활동 로깅을 설정합니다.
- AWS 암호화 솔루션을 AWS 서비스내의 모든 기본 보안 컨트롤과 함께 사용합니다.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용하세요.

- 명령줄 인터페이스 또는 API를 통해 AWS에 액세스할 때 FIPS 140-2 검증된 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 사용합니다. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 Federal Information Processing Standard(FIPS) 140-2를 참조하세요.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 양식 텍스트 필드에 입력하지 않는 것이 좋습니다. 여기에는 Device Farm 또는 기타 AWS 서비스 서비스에서 콘솔, API, AWS CLI 혹은 AWS SDK를 사용하여 작업하는 경우가 포함됩니다. 이름에 사용되는 태그 또는 자유 형식 텍스트 필드에 입력하는 모든 데이터는 청구 또는 진단 로그에 사용될 수 있습니다. 외부 서버에 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 보안 인증 정보를 URL에 포함시켜서는 안 됩니다.

## 전송 중 암호화

Device Farm 엔드포인트는 달리 언급된 경우를 제외하고 서명된 HTTPS(SSL/TLS) 요청만 지원합니다. 업로드 URL을 통해 Amazon S3에서 검색되거나 배치되는 모든 콘텐츠는 SSL/TLS를 사용하여 암호화됩니다. HTTPS 요청이 AWS 내 서명에 대한 자세한 내용은 AWS 일반 참조의 AWS API 요청에 서명을 참조하세요.

테스트되는 애플리케이션에서 수행하는 모든 통신과 디바이스 테스트를 실행하는 과정에서 설치되는 모든 애플리케이션을 암호화하고 보안을 유지하는 것은 사용자의 책임입니다.

## 저장 중 암호화

Device Farm의 데스크톱 브라우저 테스트 기능은 테스트 중에 생성된 아티팩트에 대해 저장 중 암호화를 지원합니다.

Device Farm의 물리적 모바일 디바이스 테스트 데이터는 저장 중 암호화되지 않습니다.

## 데이터 보존

Device Farm의 데이터는 제한된 시간 동안 보존됩니다. 보존 기간이 만료되면 Device Farm 백업 스토리지에서 데이터가 제거되지만 모든 메타데이터(ARN, 업로드 날짜, 파일 이름 등)는 추후 사용을 위해 보존됩니다. 다음 표에는 다양한 콘텐츠 유형의 보존 기간이 나와 있습니다.

| 콘텐츠 유형 | 보존 기간(일) |
| --- | --- |
| 업로드된 애플리케이션 | 30 |

| 콘텐츠 유형 | 보존 기간(일) |
| --- | --- |
| 업로드된 테스트 패키지 | 30 |
| 로그 | 400 |
| 비디오 녹화 자료 및 기타 아티팩트 | 400 |

더 오랜 보존을 위해 콘텐츠를 아카이빙하는 것은 사용자의 책임입니다.

## 데이터 관리

Device Farm 의 데이터는 사용되는 기능에 따라 다르게 관리됩니다. 이 섹션에서는 Device Farm을 사용하는 동안과 사용한 후에 데이터를 관리하는 방법에 대해 설명합니다.

### 데스크톱 브라우저 테스트

Selenium 세션 중에 사용된 인스턴스는 저장되지 않습니다. 세션이 종료되면 브라우저 상호 작용의 결과로 생성된 모든 데이터가 삭제됩니다.

이 기능은 현재 테스트 중에 생성된 아티팩트에 대해 저장 중 암호화를 지원합니다.

### 물리적 디바이스 테스트

다음 섹션에서는 Device Farm을 사용한 후 AWS가 디바이스를 정리하거나 파쇄하기 위해 수행하는 단계에 대한 정보를 제공합니다.

Device Farm의 물리적 모바일 디바이스 테스트 데이터는 저장 시 암호화되지 않습니다.

퍼블릭 디바이스 플릿

테스트 실행이 완료되면 이 앱 설치 제거를 비롯하여 퍼블릭 디바이스 플릿의 각 디바이스에서 일련의 정리를 수행합니다. 앱 설치 제거나 다른 정리 단계를 확인할 수 없는 경우 다시 사용하기 전에 디바이스가 초기 기본값을 수신합니다.

> **Note**
>
> 경우에 따라, 특히 앱 컨텍스트 밖에서 디바이스 시스템을 이용하는 경우 세션 간에 데이터를 유지할 수 있습니다. 이 이유로 인해 그리고 각 디바이스를 사용하는 동안 발생하는 활동 로그와 비디오를 Device Farm이 캡처하기 때문에 자동 테스트 및 원격 액세스 세션 중에는 중요한

정보(예: Google 계정 또는 Apple ID), 개인 정보, 기타 보안상 중요한 세부 정보를 입력하지 않는 것이 좋습니다.

프라이빗 디바이스

프라이빗 디바이스 계약이 만료되거나 종료된 후에는 AWS 파쇄 정책에 따라 사용할 수 없도록 디바이스가 제거되며 안전하게 파쇄됩니다. 자세한 내용은 AWS Device Farm에서 프라이빗 디바이스로 작업 단원을 참조하세요.

## 키 관리

현재 Device Farm은 저장이나 전송 중인 데이터의 암호화를 위한 외부 키 관리 기능을 제공하지 않습니다.

## 인터네트워크 트래픽 개인 정보

Device Farm은 Amazon VPC 엔드포인트를 사용하여 AWS의 리소스에 연결하도록 프라이빗 디바이스에 대해서만 구성할 수 있습니다. 계정과 연결된 비공개 AWS 인프라(예: 퍼블릭 IP 주소가 없는 Amazon EC2 인스턴스)에 대한 액세스는 Amazon VPC 엔드포인트를 사용해야 합니다. VPC 엔드포인트 구성과 관계없이 Device Farm은 Device Farm 네트워크 전체의 다른 사용자로부터 트래픽을 격리합니다.

AWS 네트워크 외부의 연결은 보안이나 안전성이 보장되지 않으며, 애플리케이션에서 수행하는 모든 인터넷 연결의 보안을 유지하는 것은 사용자의 책임입니다.

# AWS Device Farm의 복원성

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다. AWS 리전은 물리적으로 분리되고 격리된 다수의 가용 영역을 제공하며 이러한 가용 영역은 짧은 지연 시간, 높은 처리량 및 높은 중복성을 갖춘 네트워크에 연결되어 있습니다. 가용 영역을 사용하면 중단 없이 영역 간에 자동으로 장애 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 다중 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 AWS 글로벌 인프라를 참조하세요.

Device Farm은 `us-west-2` 리전에서만 사용할 수 있으므로 백업 및 복구 프로세스를 구현하는 것이 좋습니다. 이 업로드된 콘텐츠의 유일한 소스가 되어서는 안 됩니다. Device Farm이 업로드된 콘텐츠의 유일한 소스가 되어서는 안 됩니다.

Device Farm은 퍼블릭 디바이스의 가용성을 보장하지 않습니다. 이러한 디바이스는 장애율 및 격리 상태와 같은 다양한 요인에 따라 퍼블릭 디바이스 풀 내부 및 외부로 이동됩니다. 퍼블릭 디바이스 풀에 있는 디바이스의 가용성에 의존하지 않는 것이 좋습니다.

# AWS Device Farm에서 인프라 보안

관리형 서비스인 AWS Device Farm은(는) AWS 글로벌 네트워크 보안으로 보호됩니다. AWS 보안 서비스와 AWS의 인프라 보호 방법에 대한 자세한 내용은 AWS 클라우드 보안을 참조하세요. 인프라 보안에 대한 모범 사례를 사용하여 AWS 환경을 설계하려면 보안 원칙 AWS Well-Architected Framework의 인프라 보호를 참조하세요.

AWS에서 게시한 API 호출을 사용하여 네트워크를 통해 Device Farm에 액세스하세요. 고객은 다음을 지원해야 합니다.

- 전송 계층 보안(TLS) TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Ephemeral Diffie-Hellman)와 같은 완전 전송 보안(PFS)이 포함된 암호 제품군 Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 요청은 액세스 키 ID 및 IAM 주체와 관련된 보안 액세스 키를 사용하여 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용하여 임시 보안 인증 정보를 생성하여 요청에 서명할 수 있습니다.

## 물리적 디바이스 테스트 시 인프라 보안

디바이스는 물리적 디바이스 테스트 시 물리적으로 분리됩니다. 네트워크 격리로 무선 네트워크를 통한 디바이스 간 통신이 차단됩니다.

퍼블릭 디바이스는 공유되며 Device Farm은 시간 경과에 따라 디바이스를 안전하게 유지하기 위해 최선의 노력을 기울입니다. 디바이스에 대한 완전한 관리자 권리를 획득하려는 시도(루팅 또는 탈옥이라고 하는 사례)와 같은 특정 작업으로 인해 퍼블릭 디바이스가 격리됩니다. 퍼블릭 디바이스는 퍼블릭 풀에서 자동으로 제거되고 수동 검토에 배치됩니다.

프라이빗 디바이스는 명시적으로 권한이 부여된 AWS 계정으로만 액세스할 수 있습니다. Device Farm은 이러한 디바이스를 다른 디바이스로부터 물리적으로 분리하여 별도의 네트워크에 보관합니다.

비공개로 관리되는 디바이스에서는 Amazon VPC 엔드포인트를 사용하여 AWS 계정 내부 및 외부 연결을 보호하도록 테스트를 구성할 수 있습니다.

## 데스크톱 브라우저 테스트 시 인프라 보안

데스크톱 브라우저 테스트 기능을 사용하는 경우 모든 테스트 세션이 서로 분리됩니다. Selenium 인스턴스는 AWS 외부의 중간 타사 없이 교차 통신할 수 없습니다.

Selenium WebDriver 컨트롤러에 대한 모든 트래픽은 `createTestGridUrl`을 사용하여 생성된 HTTPS 엔드포인트를 통해 이루어져야 합니다.

현재 데스크톱 브라우저 테스트 기능은 Amazon VPC 엔드포인트 구성을 지원하지 않습니다. 각 Device Farm 테스트 인스턴스에서 테스트하는 리소스에 안전하게 액세스할 수 있는지 사용자가 확인해야 합니다.

# Device Farm의 구성 취약성 분석 및 관리

Device Farm을 사용하면 공급업체(예: OS 공급업체, 하드웨어 공급업체 또는 통신사)에서 적극적으로 유지 관리하거나 패치하지 않는 소프트웨어를 실행할 수 있습니다. Device Farm은 소프트웨어를 최신 상태로 유지하기 위해 최선을 다하지만 잠재적으로 취약한 소프트웨어를 사용할 수 있도록 설계된 물리적 디바이스 내 소프트웨어의 특정 버전이 최신 상태임을 보장하지는 않습니다.

예를 들어 Android 4.4.2를 실행하는 장치에서 테스트를 수행하는 경우는 StageFright라고 하는 Android의 취약성에 대해 디바이스가 패치되었는지 보장하지 않습니다. 디바이스에 보안 업데이트를 제공하는 것은 디바이스의 공급업체(때로는 통신사)에 달려 있습니다. 이 취약성을 사용하는 악성 애플리케이션은 자동 격리에 의해 포착되지 않을 수 있습니다.

프라이빗 디바이스는 AWS와 사용자의 계약에 따라 유지 관리됩니다.

Device Farm은 고객 애플리케이션이 루팅이나 탈옥과 같은 행동을 하지 못하도록 최선을 다합니다. Device Farm은 격리된 디바이스를 수동으로 검토할 때까지 공용 풀에서 제거합니다.

Python wheel 및 Ruby gem과 같이 테스트에 사용하는 소프트웨어의 모든 라이브러리 또는 버전을 최신 상태로 유지하는 것은 사용자의 책임입니다. Device Farm은 테스트 라이브러리를 업데이트할 것을 권장합니다.

이러한 리소스는 테스트 종속성을 최신 상태로 유지하는 데 도움이 될 수 있습니다.

- Ruby Gem의 보안을 유지하는 방법에 대한 자세한 내용은 RubyGems 웹 사이트의 Security Practices를 참조하세요.
- Pipenv가 사용하고 Python 패키지 기관이 보증하여 종속성 그래프에서 알려진 취약성을 검사하는 안전 패키지에 대한 자세한 내용은 GitHub의 보안 취약성 탐지를 참조하세요.

- 오픈 웹 애플리케이션 보안 프로젝트(OWASP) Maven 종속성 검사기에 대한 자세한 내용은 OWASP 웹 사이트의 OWASP DependencyCheck을 참조하세요.

자동화된 시스템이 알려진 보안 문제가 없는 것으로 판단하더라도 보안 문제가 있을 수 있음을 기억하는 것이 중요합니다. 타사의 라이브러리 또는 도구를 사용할 때는 항상 상당한 주의를 기울이고 가능하거나 필요한 경우 암호화 서명을 확인하세요.

## Device Farm에서의 사고 대응

Device Farm은 보안 문제를 나타낼 수 있는 동작이 있는지 디바이스를 지속적으로 모니터링합니다. 테스트 결과, 공용 디바이스에 기록된 파일과 같은 고객 데이터에 다른 고객이 액세스할 수 있다는 것을 AWS에서 인지할 경우 AWS는 AWS 서비스 전반에 걸쳐 사용되는 표준 인시던트 알림 및 보고 정책에 영향을 받는 고객에게 연락합니다.

## Device Farm에서 로깅 및 모니터링

이 서비스는 AWS 계정에 대한 AWS 호출을 기록하고 Amazon S3 버킷에 로그 파일을 전달하는 서비스인 AWS CloudTrail을 지원합니다. CloudTrail에서 수집된 정보를 사용하여 AWS 서비스에 대한 성공적인 요청, 요청자, 요청 시기 등을 결정할 수 있습니다. 설정 방법 및 로그 파일을 찾는 방법을 비롯한 CloudTrail에 대한 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하세요.

CodeDeploy에서 CloudTrail 사용에 관한 자세한 내용은 AWS CloudTrail을 사용하여 AWS Device Farm API 호출 로깅 단원을 참조하세요.

## 디바이스 에이전트의 보안 모범 사례

Device Farm은 자체 보안 정책을 개발하고 구현할 때 고려해야 할 여러 보안 기능을 제공합니다. 다음 모범 사례는 일반적인 지침이며 완벽한 보안 솔루션을 나타내지는 않습니다. 이러한 모범 사례는 환경에 적절하지 않거나 충분하지 않을 수 있으므로 참고용으로만 사용해 주세요.

- 사용하는 지속적 통합(CI) 시스템에 IAM에서 가능한 최소 권한을 부여합니다. CI 시스템이 손상될 경우에도 허위 요청을 할 수 없도록 각 CI 시스템 테스트에 임시 보안 인증을 사용하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 임시 보안 인증을 참조하세요.
- 사용자 지정 테스트 환경에서 adb 명령을 사용하여 애플리케이션에서 생성된 콘텐츠를 정리합니다. 사용자 지정 테스트 환경에 대한 자세한 내용은 사용자 지정 테스트 환경 작업 단원을 참조하세요.

# AWS Device Farm에 사용되는 제한

다음 목록에서는 현재 AWS Device Farm 제한을 설명합니다.

- 업로드할 수 있는 앱의 최대 파일 크기는 4GB입니다.
- 테스트 실행에 포함할 수 있는 디바이스 수에는 제한이 없습니다. 그러나 테스트 실행 중 Device Farm이 동시에 테스트할 최대 디바이스 수는 5개입니다. (이 숫자는 요청에 따라 늘릴 수 있습니다.)
- 예약할 수 있는 실행 수에는 제한이 없습니다.
- 원격 액세스 세션의 기간 제한은 150분입니다.
- 자동 테스트 실행의 기간 제한은 150분입니다.
- 계정 전체에서 대기 중인 작업을 포함하여 진행 중인 작업의 최대 수는 250개입니다. 이것은 소프트 제한입니다.
- 테스트 실행에 포함할 수 있는 디바이스 수에는 제한이 없습니다. 언제든지 병렬로 테스트를 실행할 수 있는 디바이스 또는 작업의 수는 계정 수준의 동시성과 동일합니다. AWS Device Farm에서 제한 사용에 대한 기본 계정 수준 동시성은 5입니다. 사용 사례에 따라 이 수치를 특정 임계값까지 늘리도록 요청할 수 있습니다. 무제한 사용에 대한 기본 계정 수준 동시성은 해당 플랫폼에서 구독하는 슬롯 수와 동일합니다.

# AWS Device Farm에 사용되는 도구와 플러그인

이 섹션은 AWS Device Farm 도구 및 플러그인 사용에 대한 링크와 정보를 포함합니다. Device Farm 플러그인은 GitHub의 AWS 랩에서 찾을 수 있습니다.

Android 개발자라면 GitHub의 Android용 AWS Device Farm 샘플 앱도 이용할 수 있습니다. 자체 Device Farm 테스트 스크립트을 위한 레퍼런스로 앱과 예제 테스트를 사용할 수 있습니다.

주제

- AWS Device Farm과 Jenkins CI 플러그인 통합
- AWS Device Farm Gradle 플러그인

## AWS Device Farm과 Jenkins CI 플러그인 통합

이 플러그인은 자체 Jenkins 연속 통합(CI) 서버에서 AWS Device Farm 기능을 제공합니다. 자세한 내용은 Jenkin(소프트웨어)를 참조하세요.

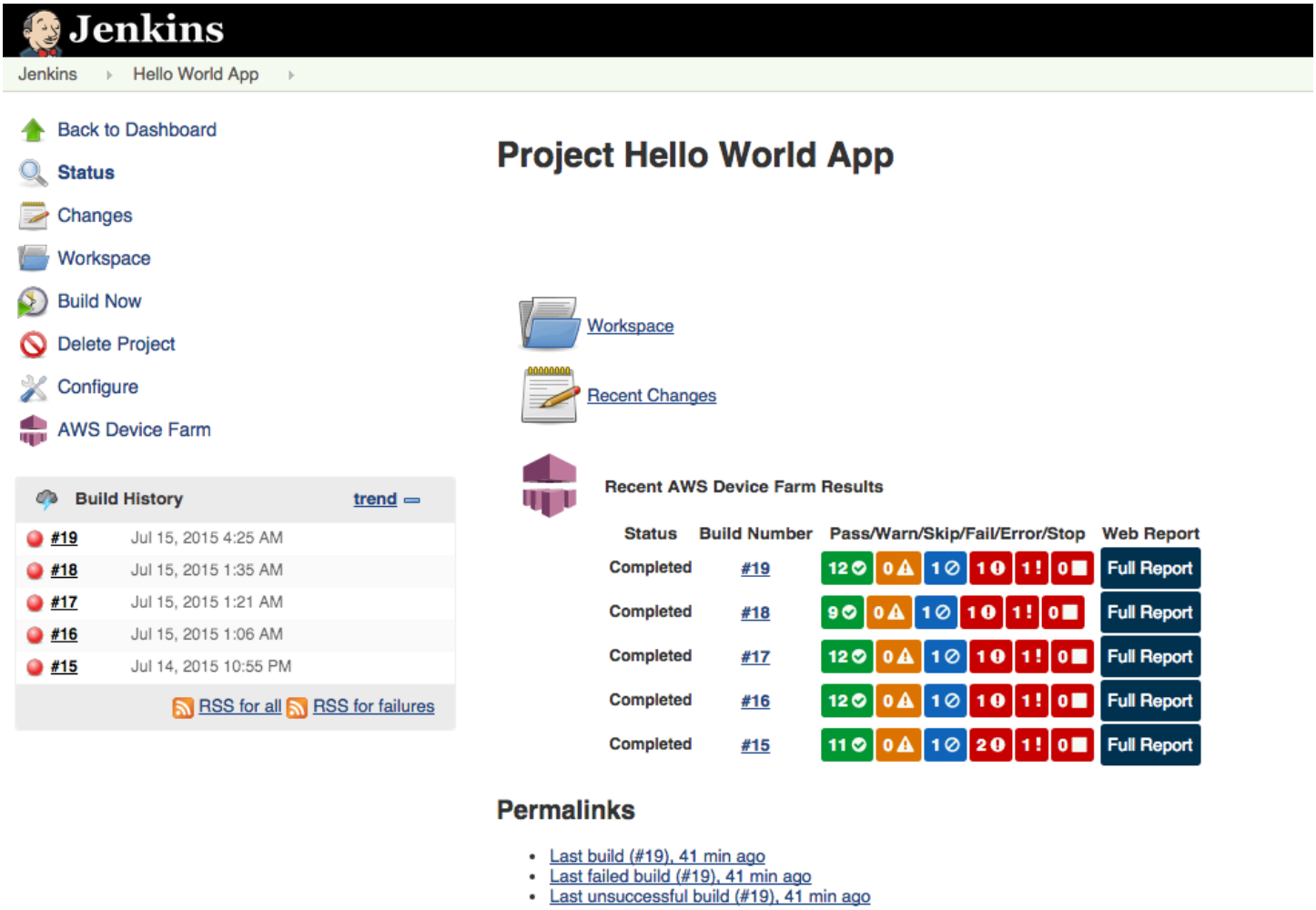
> **Note**
>
> Jenkins 플러그인을 다운로드하려면 GitHub로 이동하여 1단계: 플러그인 설치 지침을 따르세요.

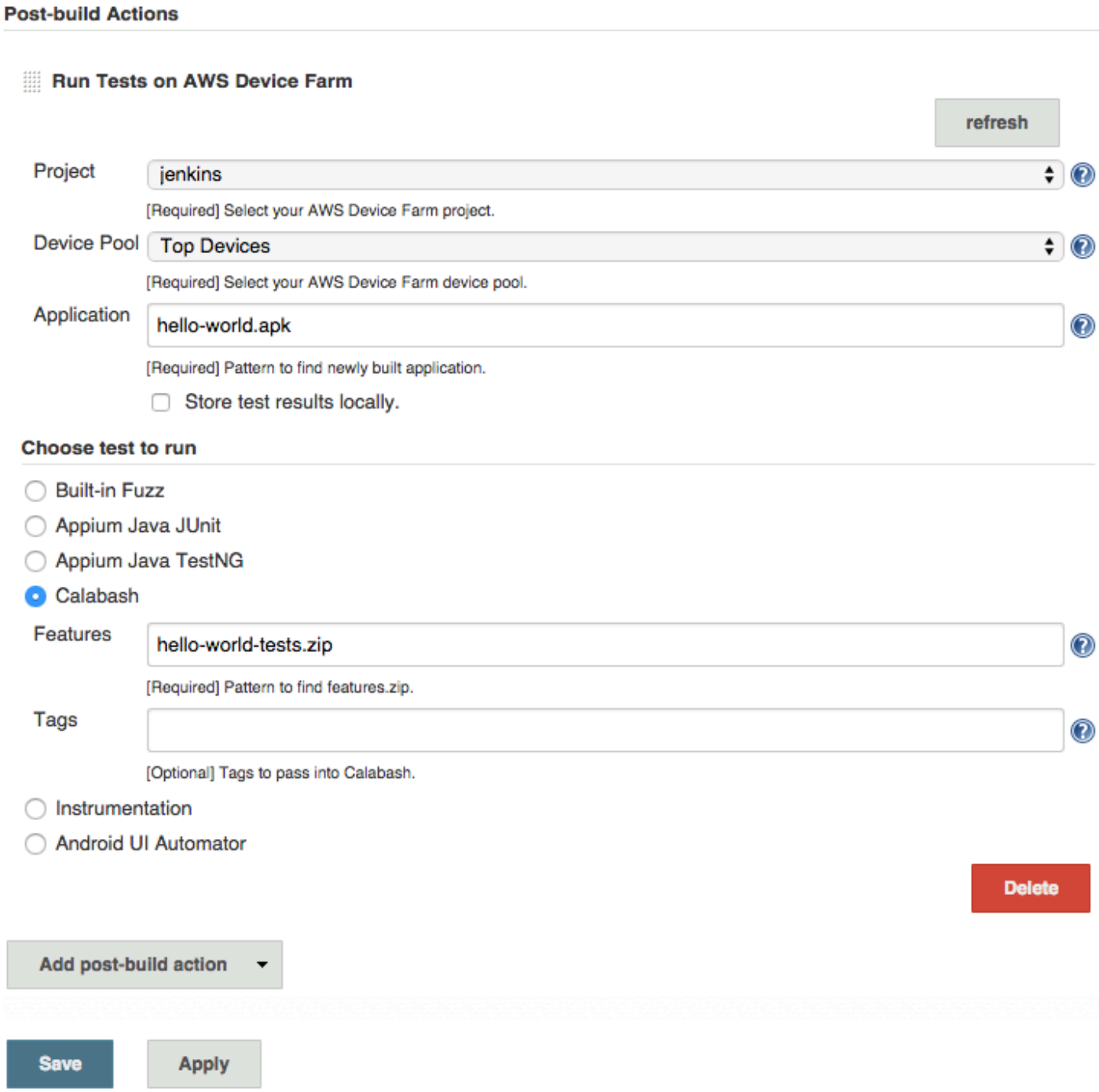
이 섹션에는 AWS Device Farm에서 Jenkins CI 플러그인을 설정하고 사용하기 위한 일련의 절차가 포함되어 있습니다.

주제

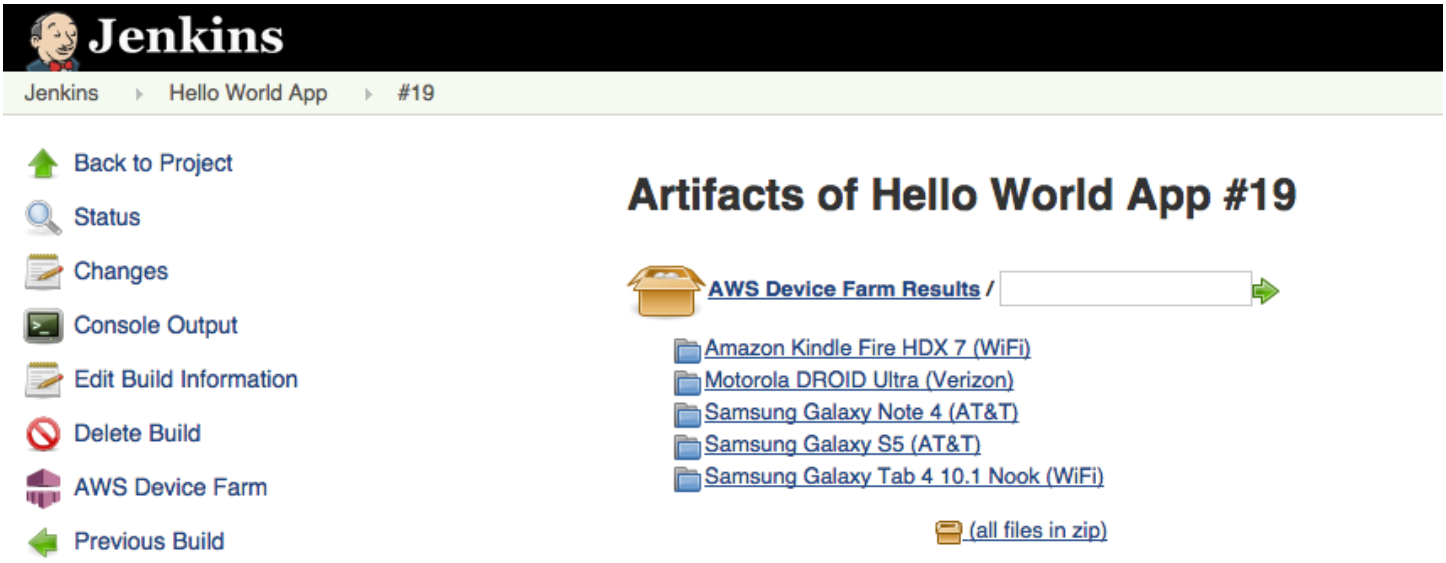
- 1단계: 플러그인 설치
- 2단계: Jenkins CI 플러그인을 위한 AWS Identity and Access Management 사용자 생성
- 3단계: 최초 구성 지침
- 4단계: Jenkins 작업에서 플러그인 사용
- 종속성

다음 이미지는 Jenkins CI 플러그인의 기능을 보여줍니다.


Jenkins
Jenkins ▸ Hello World App ▸
Back to Dashboard
Status
Changes
Workspace
Build Now
Delete Project
Configure
AWS Device Farm
Build History trend
#19 Jul 15, 2015 4:25 AM
#18 Jul 15, 2015 1:35 AM
#17 Jul 15, 2015 1:21 AM
#16 Jul 15, 2015 1:06 AM
#15 Jul 14, 2015 10:55 PM
RSS for all RSS for failures
Project Hello World App
Workspace
Recent Changes
Recent AWS Device Farm Results
Status Build Number Pass/Warn/Skip/Fail/Error/Stop Web Report
Completed #19 12 0 1 1 1 0 Full Report
Completed #18 9 0 1 1 1 0 Full Report
Completed #17 12 0 1 1 1 0 Full Report
Completed #16 12 0 1 1 1 0 Full Report
Completed #15 11 0 1 2 1 0 Full Report
Permalinks
Last build (#19), 41 min ago
Last failed build (#19), 41 min ago
Last unsuccessful build (#19), 41 min ago




플러그인은 모든 테스트 아티팩트(로그, 스크린샷 등) 또한 로컬에서 가져올 수 있습니다.




## 1단계: 플러그인 설치

AWS Device Farm용 Jenkins 지속적 통합(CI) 플러그인을 설치하는 데는 두 가지 옵션이 있습니다. Jenkins 웹 UI의 사용 가능한 플러그인 대화 상자에서 플러그인을 검색하거나 Jenkins 내에서 `hpi` 파일을 다운로드하여 설치할 수 있습니다.

### 젠킨스 UI 내에서 설치

1. Jenkins 관리, 플러그인 관리를 선택한 다음 가용성을 선택하여 Jenkins UI에서 플러그인을 찾으세요.
2. aws-device-farm을 검색하세요.
3. AWS Device Farm 플러그인을 설치하세요.
4. 플러그인이 `Jenkins` 사용자 소유인지 확인하세요.
5. Jenkins 다시 시작

### 플러그인 다운로드

1. http://updates.jenkins-ci.org/latest/aws-device farm.hpi에서 직접 `hpi` 파일을 다운로드하세요.
2. 플러그인이 `Jenkins` 사용자 소유인지 확인하세요.
3. 다음 옵션 중 하나를 사용하여 플러그인을 설치하세요.

- Jenkins 관리, 플러그인 관리, 고급을 선택하여 플러그인을 업로드한 다음 플러그인 업로드를 선택하세요.
- hpi 파일을 Jenkins 플러그인 디렉터리(보통 `/var/lib/jenkins/plugins`)에 넣습니다.

4. Jenkins를 다시 시작하세요.

## 2단계: Jenkins CI 플러그인을 위한 AWS Identity and Access Management 사용자 생성

Device Farm에 액세스하기 위해 AWS 루트 계정을 사용하지 않는 것이 좋습니다. 대신 AWS 계정에 AWS Identity and Access Management(IAM) 사용자를 만들거나 기존 IAM 사용자를 사용한 다음, 해당 IAM 사용자를 통해 Device Farm에 액세스하세요.

새 IAM 사용자를 생성하려면 IAM 사용자 생성(AWS Management Console)을 참조하세요. 각 사용자에 대한 액세스 키를 생성하고 사용자 보안 인증 정보를 다운로드하거나 저장해야 합니다. 나중에 보안 인증이 필요합니다.

### IAM 사용자에게 Device Farm에 액세스할 수 있는 권한 부여

IAM 사용자에게 Device Farm에 액세스할 수 있는 권한을 부여하려면 IAM에서 새 액세스 정책을 생성하고 다음과 같이 IAM 사용자에게 액세스 정책을 할당하세요.

> **Note**
>
> 다음 단계를 완료하기 위해 사용하는 IAM 사용자 또는 AWS 루트 계정은 다음 IAM 정책을 만들고 IAM 사용자에게 연결할 수 있는 권한을 가지고 있어야 합니다. 자세한 내용은 정책 사용을 참조하세요.

IAM에서 액세스 정책을 생성하려면 다음을 참조하세요.

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 여세요.
2. 정책을 선택하세요.
3. 정책 생성을 선택하세요. 지금 시작 버튼이 표시되면 선택한 후 정책 생성을 선택하세요.
4. 자체 정책 생성 옆의 선택을 선택하세요.
5. 정책 이름에 정책 이름을 입력하세요(예: **`AWSDeviceFarmAccessPolicy`**).
6. 설명에는 이 IAM 사용자를 Jenkins 프로젝트와 연결하는 데 도움이 되는 설명을 입력하세요.

7. 정책 문서에 다음 설명을 입력하세요.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DeviceFarmAll",
            "Effect": "Allow",
            "Action": [ "devicefarm:*" ],
            "Resource": [ "*" ]
        }
    ]
}
```

8. 정책 생성을 선택하세요.

IAM 사용자에게 액세스 정책을 할당하려면 다음을 참조하세요.

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 여세요.
2. 사용자를 선택하세요.
3. 액세스 정책을 할당하려는 IAM 사용자를 선택하세요.
4. 권한 영역의 관리형 정책에서 정책 연결을 선택하세요.
5. 방금 생성한 정책(예: AWSDeviceFarmAccessPolicy)을 선택하세요.
6. 정책 연결을 선택하세요.

## 3단계: 최초 구성 지침

Jenkins 서버를 처음 실행할 때는 다음과 같이 시스템을 구성해야 합니다.

> **Note**
>
> 디바이스 슬롯을 사용하는 경우 디바이스 슬롯 기능은 기본적으로 비활성화됩니다.

1. Jenkins 웹 사용자 인터페이스에 로그인하세요.
2. 화면 왼쪽에서 Jenkins 관리를 선택하세요.
3. 시스템 구성을 선택하세요.

4. 아래로 스크롤하여 AWS Device Farm 헤더로 이동하세요.
5. 2단계: IAM 사용자 생성에서 보안 인증 정보를 복사하여 액세스 키 ID 및 비밀 액세스 키를 각각 해당 상자에 붙여 넣으세요.
6. 저장을 선택하세요.

## 4단계: Jenkins 작업에서 플러그인 사용

Jenkins 플러그인을 설치했으면 다음 지침에 따라 Jenkins 작업에서 플러그인을 사용하세요.

1. Jenkins 웹 UI에 로그인하세요.
2. 편집할 작업을 클릭하세요.
3. 화면 왼쪽에서 구성을 선택하세요.
4. 아래로 스크롤하여 빌드 후 작업 헤더로 이동하세요.
5. 빌드 후 작업 추가를 클릭하고 AWS Device Farm에서 테스트 실행을 선택하세요.
6. 사용하려는 프로젝트를 선택하세요.
7. 사용할 디바이스 풀을 선택하세요.
8. 테스트 아티팩트(예: 로그 및 스크린샷)를 로컬에 보관할지 여부를 선택하세요.
9. 애플리케이션에서 컴파일된 애플리케이션의 경로를 입력하세요.
10. 실행할 테스트를 선택하고 필수 필드를 모두 입력하세요.
11. 저장을 선택하세요.

## 종속성

Jenkins CI 플러그인을 사용하려면 AWS 모바일 SDK 1.10.5 이상이 필요합니다. 자세한 내용과 SDK를 설치하려면 AWS Mobile SDK를 참조하세요.

# AWS Device Farm Gradle 플러그인

이 플러그인은 Android Studio의 Gradle 빌드 시스템과 AWS Device Farm을 통합합니다. 자세한 내용은 Gradle을 참조하세요.

> **Note**
>
> Gradle 플러그인을 다운로드하려면 GitHub로 이동하여 Device Farm Gradle 플러그인 구축 내 설명을 따르세요.

Device Farm Gradle 플러그인은 Android Studio 환경에서 Device Farm이 기능하도록 합니다. Device Farm이 호스팅하는 테스트를 실제 Android 휴대폰 및 태블릿에서 시작할 수 있습니다.

이 단원은 Device Farm Gradle 플러그인을 설정하고 사용하는 여러 절차를 포함합니다.

주제



## 1단계: AWS Device Farm Gradle 플러그인 구축

이 플러그인은 Android Studio의 Gradle 빌드 시스템과 AWS Device Farm을 통합합니다. 자세한 내용은 Gradle을 참조하세요.

> **Note**
>
> 플러그인 구축은 선택 사항입니다. 플러그인은 Maven Central에 게시됩니다. Gradle에서 직접 플러그인을 다운로드하려면 이 단계를 건너뛰고 2단계: AWS Device Farm Gradle 플러그인 설정로 이동하세요.

플러그인 구축

1. GitHub에서 리포지토리를 복제하세요.
2. `gradle install`을 사용하여 플러그인을 구축하세요.

   플러그인은 로컬 Maven 리포지토리에 설치됩니다.

다음 단계: 2단계: AWS Device Farm Gradle 플러그인 설정

# 2단계: AWS Device Farm Gradle 플러그인 설정

아직 리포지토리를 복제하고 플러그인을 설치하지 않았다면 Device Farm Gradle 플러그인 구축 절차를 따르세요.

AWS Device Farm Gradle 플러그인 구성

1. `build.gradle`의 종속성 목록에 플러그인 아티팩트를 추가하세요.

```
    buildscript {

        repositories {
            mavenLocal()
            mavenCentral()
        }

        dependencies {
            classpath 'com.android.tools.build:gradle:1.3.0'
            classpath 'com.amazonaws:aws-devicefarm-gradle-plugin:1.0'
        }
    }
```

2. `build.gradle` 파일에 플러그인을 구성하세요. 다음에 제시되는 테스트별 구성이 안내서 역할을 합니다.

```
apply plugin: 'devicefarm'

devicefarm {

    // Required. The project must already exist. You can create a project in the
 AWS Device Farm console.
    projectName "My Project" // required: Must already exist.

    // Optional. Defaults to "Top Devices"
    // devicePool "My Device Pool Name"

    // Optional. Default is 150 minutes
    // executionTimeoutMinutes 150

    // Optional. Set to "off" if you want to disable device video recording during
 a run. Default is "on"
```

```
   // videoRecording "on"

   // Optional. Set to "off" if you want to disable device performance monitoring
during a run. Default is "on"
   // performanceMonitoring "on"

   // Optional. Add this if you have a subscription and want to use your unmetered
slots
   // useUnmeteredDevices()

   // Required. You must specify either accessKey and secretKey OR roleArn.
roleArn takes precedence.
   authentication {
       accessKey "AKIAIOSFODNN7EXAMPLE"
       secretKey "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"

       // OR

       roleArn "arn:aws:iam::111122223333:role/DeviceFarmRole"
   }

   // Optionally, you can
   // - enable or disable Wi-Fi, Bluetooth, GPS, NFC radios
   // - set the GPS coordinates
   // - specify files and applications that must be on the device when your test
runs
   devicestate {
       // Extra files to include on the device.
       // extraDataZipFile file("path/to/zip")

       // Other applications that must be installed in addition to yours.
       // auxiliaryApps files(file("path/to/app"), file("path/to/app2"))

       // By default, Wi-Fi, Bluetooth, GPS, and NFC are turned on.
       // wifi "off"
       // bluetooth "off"
       // gps "off"
       // nfc "off"

       // You can specify GPS location. By default, this location is 47.6204,
-122.3491
       // latitude 44.97005
       // longitude -93.28872
   }
```

```
    // By default, the Instrumentation test is used.
    // If you want to use a different test type, configure it here.
    // You can set only one test type (for example, Calabash, Fuzz, and so on)

    // Fuzz
    // fuzz { }

    // Calabash
    // calabash { tests file("path-to-features.zip") }

}
```

3. `gradle devicefarmUpload`의 태스크를 사용하여 Device Farm 테스트를 실행하세요.

   빌드 출력은 테스트 실행 모니터링이 가능한 Device Farm 콘솔로의 링크를 출력합니다.

다음 단계: IAM 사용자 생성

## 3단계: IAM 사용자 생성

AWS Identity and Access Management(IAM)는 AWS 리소스 작업에 대한 권한 및 정책을 관리를 돕습니다. 여기에서는 AWS Device Farm 리소스에 액세스할 수 있는 권한을 가진 IAM 사용자 생성 방법을 안내합니다.

IAM 사용자를 설치하지 않았다면 1단계와 2단계를 먼저 수행하세요.

Device Farm에 액세스하기 위해 AWS 루트 계정을 사용하지 않는 것이 좋습니다. 대신에 AWS 계정에서 새로운 IAM 사용자를 생성(혹은 기존 IAM 사용자 사용)한 다음, 해당 사용자로 Device Farm에 액세스합니다.

> **Note**
>
> 다음 단계를 완료하기 위해 사용하는 AWS 루트 계정 혹은 IAM 사용자가 다음 IAM 정책을 만들고 사용자에게 연결할 수 있는 권한을 가져야 합니다. 자세한 내용은 정책 작업을 참조하세요.

IAM 액세스 정책에 맞추어 새 사용자 생성

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 여세요.
2. 사용자를 선택하세요.
3. 새 사용자 생성을 선택하세요.
4. 선택한 사용자 이름을 입력하세요.

   예시: **`GradleUser`**
5. 생성을 선택하세요.
6. 자격 증명 다운로드를 선택하고 나중에 쉽게 찾을 수 있는 위치에 저장하세요.
7. 닫기를 선택하세요.
8. 목록에서 사용자 이름을 선택하세요.
9. 권한에서 오른쪽의 아래쪽 화살표를 선택하여 인라인 정책 헤더를 펼치세요.
10. 표시할 인라인 정책이 없습니다가 표시된 여기를 클릭을 선택하세요. 생성하려면 여기를 클릭하세요.
11. 권한 설정 화면에서 사용자 지정 정책을 선택하세요.
12. 선택을 선택하세요.
13. 정책에 이름을 지정하세요(예시: **`AWSDeviceFarmGradlePolicy`**).
14. 정책 문서에 다음 정책을 붙여 넣으세요.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DeviceFarmAll",
            "Effect": "Allow",
            "Action": [ "devicefarm:*" ],
            "Resource": [ "*" ]
        }
    ]
}
```

15. 정책 적용을 선택하세요.

다음 단계: 테스트 유형 구성

자세한 내용은 IAM 사용자 생성(AWS Management Console) 또는 설정 단원을 참조하세요.

# 4단계: 테스트 유형 구성

기본적으로 AWS Device Farm Gradle 플러그인이 Android 및 AWS Device Farm용 계측 작업 테스트를 실행합니다. 자체 테스트를 실행하거나 추가 파라미터를 지정하는 경우 테스트 유형을 구성할 수 있습니다. 여기에서는 사용 가능한 각 테스트 유형과 구성을 위해 Android Studio에서 수행해야 작업을 설명합니다. Device Farm에서 사용할 수 있는 테스트 유형에 대한 자세한 내용은 AWS Device Farm에서 테스트 유형을 사용한 작업 단원을 참조하세요.

테스트 유형을 구성하기 전이라면 먼저 1~3단계를 완료하세요.

> **Note**
> 디바이스 슬롯을 사용하는 경우 해당 기능은 기본적으로 비활성화 상태입니다.

## Appium

Device Farm은 Android용 Appium Java JUnit과 TestNG를 지원합니다.

- Appium(Java(JUnit) 환경)
- Appium(Java(TestNG) 환경)

`useTestNG()` 또는 `useJUnit()`을 선택할 수 있습니다. 기본값은 `JUnit`이며 명시적으로 지정할 필요는 없습니다.

```
    appium {
        tests file("path to zip file") // required
        useTestNG() // or useJUnit()
    }
```

## 내장: fuzz

Device Farm은 장치에 임의로 사용자 인터페이스 이벤트를 전송하고 결과를 보고하는 내장 fuzz 테스트 유형을 제공합니다.

```
    fuzz {

       eventThrottle 50 // optional default
       eventCount 6000  // optional default
```

```
        randomizerSeed 1234 // optional default blank

    }
```

자세한 설명은 내장: fuzz (Android 및 iOS) 섹션을 참조하세요.

### 계측

Device Farm은 Android의 계측(JUnit, Espresso, Robotium 및 모든 계측 기반 테스트)을 지원합니다. 자세한 내용은 Android 및 AWS Device Farm용 계측 작업 단원을 참조하세요.

Gradle에서 계측 테스트를 실행할 때 Device Farm은 androidTest 디렉터리에서 생성된 `.apk` 파일을 테스트의 소스로 사용합니다.

```
    instrumentation {

        filter "test filter per developer docs" // optional

    }
```

## 의존성

### 런타임

- Device Farm Gradle 플러그인을 사용하려면 AWS 모바일 SDK 1.10.15 이상이 필요합니다. 자세한 내용 및 SDK 설치를 원한다면 AWS Mobile SDK를 참조하세요.
- Android tools builder test api 0.5.2
- Apache Commons Lang3 3.3.4

### 유닛 테스트의 경우

- Testng 6.8.8
- Jmocit 1.19
- Android Gradle Tools 1.3.0

# 문서 기록

다음 표에서는 이 가이드의 최신 릴리스가 발표된 이후 이 설명서에서 변경된 중요 사항에 대해 설명합니다.

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| AL2 지원 | Device Farm은 Android용 AL2 테스트 환경을 지원합니다. AL2에 대해 자세히 알아보십시오. | 2023년 11월 6일 |
| 표준 테스트 환경에서 사용자 지정 테스트 환경으로 마이그레이션 | 2023년 12월 표준 모드 테스트의 사용 중단을 문서화하도록 마이그레이션 안내서가 업데이트되었습니다. | 2023년 9월 3일 |
| VPC ENI 지원 | 이제 Device Farm을 통해 프라이빗 디바이스가 VPC-ENI 연결 기능을 사용하여 고객이 AWS 온프레미스 소프트웨어 또는 다른 클라우드 제공업체에서 호스팅된 프라이빗 엔드포인트에 안전하게 연결하게 할 수 있습니다. VPC-ENI에 대해 자세히 알아보십시오. | 2023년 5월 15일 |
| Polaris UI 업데이트 | Device Farm 콘솔은 이제 Polaris 프레임워크를 지원합니다. | 2021년 7월 28일 |
| Python 3 지원 | Device Farm은 이제 사용자 지정 모드 테스트에서 Python 3를 지원합니다. 테스트 패키지에서 Python 3을 사용하는 방법에 대해 자세히 알아보세요.<br>• Appium (Python)<br>• Appium (Python) | 2020년 4월 20일 |
| 새로운 보안 정보 및 AWS 리소스 태그 지정 정보 | 보다 쉽고 포괄적으로 AWS 서비스의 보안을 유지할 수 있도록 하기 위해 보안에 대한 새로운 단원이 작성되었습니다. 자세한 내용은 AWS Device Farm의 보안 단원을 참조하세요. | 2020년 3월 27일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| | Device Farm의 태그 지정에 대한 새 단원이 추가되었습니다. 태그 지정에 대한 자세한 내용은 Device Farm에 태그 지정 단원을 참조하세요. | |
| 다이렉트 디바이스 액세스 제거. | 다이렉트 디바이스 액세스(프라이빗 디바이스에 대한 원격 디버깅)를 더 이상 일반 사용 사례에 사용할 수 없습니다. 다이렉트 디바이스 액세스의 향후 가용성에 대해 문의하려면 문의하기하세요. | 2019년 9월 9일 |
| Gradle 플러그인 구성 업데이트 | 이제 개정된 Gradle 플러그인 구성에 선택적 파라미터가 코멘트 아웃 처리된 사용자 지정 가능한 버전의 Gradle 구성이 포함됩니다. Device Farm Gradle 플러그인 설정 단원에 대해 자세히 알아보세요. | 2019년 8월 16일 |
| XCTest로 테스트 실행을 위한 새 요구 사항 | XCTest 프레임워크를 사용하는 테스트 실행의 경우 Device Farm을 사용하려면 이제 테스트용으로 빌드된 앱 패키지가 필요합니다. the section called “XCTest” 단원에 대해 자세히 알아보세요. | 2019년 2월 4일 |
| 사용자 지정 환경에서 Appium Node.js 및 Appium Ruby 테스트 유형을 지원합니다. | Appium Node.js 및 Appium Ruby 사용자 지정 테스트 환경에서 테스트를 실행할 수 있게 되었습니다. AWS Device Farm에서 테스트 유형을 사용한 작업 단원에 대해 자세히 알아보세요. | 2019년 1월 10일 |
| 표준 및 사용자 지정 환경 모두에서 Appium 서버 버전 1.7.2를 지원합니다. 사용자 지정 테스트 환경에서 사용자 지정 테스트 사양 YAML 파일을 사용하여 버전 1.8.1을 지원합니다. | 이제 Appium 서버 버전 1.72, 1.71, 1.6.5가 설치된 표준 및 사용자 지정 테스트 환경 모두에서 테스트를 실행할 수 있습니다. 또한 사용자 지정 테스트 환경에서 사용자 지정 테스트 사양 YAML 파일을 사용하여 버전 1.8.1 및 1.8.0으로 테스트를 실행할 수도 있습니다. AWS Device Farm에서 테스트 유형을 사용한 작업 단원에 대해 자세히 알아보세요. | 2018년 10월 2일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| 사용자 지정 테스트 환경 | 사용자 지정 테스트 환경을 사용하면 로컬 환경에서처럼 테스트를 실행할 수 있습니다. 이제 Device Farm은 라이브 로그 및 비디오 스트리밍을 지원하므로 사용자 지정 테스트 환경에서 실행되는 테스트에 대한 피드백을 즉시 받을 수 있습니다. 사용자 지정 테스트 환경 작업 단원에 대해 자세히 알아보세요. | 2018년 8월 16일 |
| AWS CodePipeline 테스트 공급자로 Device Farm 사용 지원 | 이제 릴리스 프로세스에서 AWS CodePipeline AWS Device Farm 실행을 테스트 작업으로 사용하도록 파이프라인을 구성할 수 있습니다. CodePipeline 리포지토리를 구축 및 테스트 단계에 빠르게 연결하여 필요에 맞게 사용자 지정된 지속적 통합 시스템을 구축할 수 있습니다. CodePipeline 테스트 단계에서 AWS Device Farm 사용 단원에 대해 자세히 알아보세요. | 2018년 7월 19일 |
| 프라이빗 디바이스 지원 | 이제 프라이빗 디바이스를 사용하여 테스트 실행을 예약하고 원격 액세스 세션을 시작할 수 있습니다. 이 디바이스의 프로필과 설정을 관리하고, Amazon VPC 엔드포인트를 생성하여 프라이빗 앱을 테스트하며, 원격 디버깅 세션을 생성할 수 있습니다. AWS Device Farm에서 프라이빗 디바이스로 작업 단원에 대해 자세히 알아보세요. | 2018년 5월 2일 |
| Appium 1.6.3 지원 | 이제 Appium 사용자 지정 테스트용 Appium 버전을 설정할 수 있습니다. | 2017년 3월 21일 |
| 테스트 실행을 위한 실행 제한 시간 설정 | 테스트 실행 또는 프로젝트의 모든 테스트에 대해 실행 제한 시간을 설정할 수 있습니다. AWS Device Farm의 테스트 실행 시간 제한 설정 단원에 대해 자세히 알아보세요. | 2017년 2월 9일 |
| 네트워크 셰이핑 | 이제 테스트 실행을 위한 네트워크 연결 및 조건을 시뮬레이션할 수 있습니다. AWS Device Farm 실행을 위한 네트워크 연결 및 조건 시뮬레이션 단원에 대해 자세히 알아보세요. | 2016년 12월 8일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| 새로운 문제 해결 섹션 | 이제 Device Farm 콘솔에서 발생할 수 있는 오류 메시지를 해결하는 절차를 통해 테스트 패키지 업로드 문제를 해결할 수 있습니다. Device Farm 오류 문제 해결 단원에 대해 자세히 알아보세요. | 2016년 8월 10일 |
| 원격 액세스 세션 | 이제 콘솔에서 단일 디바이스에 원격으로 액세스하고 상호 작용할 수 있습니다. 원격 액세스 작업 단원에 대해 자세히 알아보세요. | 2016년 4월 19일 |
| 디바이스 슬롯 셀프 서비스 | 이제 AWS Management Console, AWS Command Line Interface 또는 API를 사용하여 디바이스 슬롯을 구입할 수 있습니다. Device Farm에서 디바이스 슬롯 구매 방법에 대해 자세히 알아보세요. | 2016년 3월 22일 |
| 테스트 실행을 중지하는 방법 | 이제 AWS Management Console, AWS Command Line Interface 또는 API를 사용하여 테스트 실행을 중지할 수 있습니다. AWS Device Farm에서의 실행 중지 방법에 대해 자세히 알아보세요. | 2016년 3월 22일 |
| 새로운 XCTest UI 테스트 유형 | 이제 iOS 애플리케이션에서 XCTest UI 사용자 지정 테스트를 실행할 수 있습니다. XCTest UI 테스트 유형에 대해 자세히 알아보세요. | 2016년 3월 8일 |
| 새로운 Appium Python 테스트 유형 | 이제 Android, iOS 및 웹 애플리케이션에서 Appium Python 사용자 지정 테스트를 실행할 수 있습니다. AWS Device Farm에서 테스트 유형을 사용한 작업 단원에 대해 자세히 알아보세요. | 2016년 1월 19일 |
| 웹 애플리케이션 테스트 유형 | 이제 웹 애플리케이션에서 Appium Java JUnit 및 TestNG 사용자 지정 테스트를 실행할 수 있습니다. AWS Device Farm에서 웹 앱 테스트 사용 단원에 대해 자세히 알아보세요. | 2015년 11월 19일 |
| AWS Device Farm Gradle 플러그인 | Device Farm Gradle 플러그인 설치 및 사용 방법에 대해 자세히 알아보세요. | 2015년 9월 28일 |

| 변경 사항 | 설명 | 변경 날짜 |
| --- | --- | --- |
| 새로운 Android 기본 제공 테스트: 탐색기 | 탐색기 테스트는 마치 최종 사용자인 것처럼 각 화면을 분석하여 앱을 크롤링하고, 탐색하는 동안 스크린샷을 캡처합니다. | 2015년 9월 16일 |
| iOS 지원 추가됨 | AWS Device Farm에서 테스트 유형을 사용한 작업에서 iOS 디바이스 테스트 및 iOS 테스트(XCTest 포함) 실행에 대해 자세히 알아보세요. | 2015년 8월 4일 |
| 최초 공개 릴리스 | AWS Device Farm 개발자 안내서가 처음으로 릴리스되었습니다. | 2015년 7월 13일 |

# AWS 용어집

최신 AWS 용어는 AWS 용어집 참조의 AWS 용어집을 참조하세요.

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.