Entwicklerhandbuch

AWS Device Farm

API-Version 2015-06-23

AWS Device Farm: Entwicklerhandbuch

Table of Contents

Was ist AWS Device Farm?

Device Farm ist ein App-Testservice, mit dem Sie Ihre Android-, iOS- und Web-Apps auf echten, physischen Telefonen und Tablets testen und mit ihnen interagieren können, die von Amazon Web Services (AWS) gehostet werden.

Device Farm kann hauptsächlich auf zwei Arten verwendet werden:

- Greifen Sie von Ihrem lokalen Computer aus remote auf ein Gerät zu, entweder interaktiv in Ihrem Webbrowser oder testen Sie es automatisch mit Appium von einem lokalen Client aus.
- Führen Sie App-Tests mithilfe der verwalteten Testausführungsumgebung von Device Farm automatisch aus.

> **Note**
>
> Device Farm ist nur in der Region `us-west-2` (Oregon) verfügbar.

Remote-Zugriff

Durch den Fernzugriff können Sie über Ihren Webbrowser in Echtzeit mit einem Gerät interagieren. Mit dem Fernzugriff können Sie auch Appium-Tests von Ihrem lokalen Client aus auf entfernten Device Farm Farm-Geräten ausführen, die einen verwalteten Appium-Endpunkt verwenden.

Die Interaktion mit einem Gerät in Echtzeit kann für eine Reihe von Szenarien nützlich sein, z. B. für manuelles Testen von Apps, die Reproduktion von Fehlern auf einem bestimmten Gerät, die Überprüfung der visuellen Darstellung Ihrer App auf verschiedenen Bildschirmtypen sowie für die Installation und Aktualisierung von Apps. Der vollständig verwaltete Appium-Endpunkt von Device Farm ermöglicht es Ihnen, Ihre Appium-Tests zu entwickeln, zu testen und zu debuggen und so schnelles Feedback zu erhalten.

Der Appium-Endpunkt unterstützt jede Sprache Ihrer Wahl, jede lokale IDE, Live-Debugging mit Breakpoints, Live-Video und Logs sowie Tools wie Appium Inspector. Sie können während Ihrer Fernzugriffssitzung beliebig oft Tests auf demselben Gerät mit einem Limit von 150 Minuten ausführen.

Während einer Fernzugriffssitzung protokolliert Device Farm Details zu Aktionen, die während Ihrer Interaktion mit dem Gerät stattfinden. Protokolle mit diesen Informationen sowie eine Videoaufnahme der Sitzung werden am Ende der Sitzung bereitgestellt.

Automatisierte App-Tests

Device Farm ermöglicht es Ihnen, automatisierte Tests auf mehreren Geräten parallel auszuführen, indem Sie Ihre App und Tests hochladen. Die Tests werden automatisch in einer vollständig verwalteten Umgebung auf Testhosts ausgeführt, auf denen Sie eine Testspezifikationsdatei konfigurieren können. Die Umgebung verwendet die Testhosts von Device Farm, sodass Sie sich keine Gedanken über die Bereitstellung Ihrer eigenen Infrastruktur für die Ausführung von Tests machen müssen. Die Testhosts und -geräte können sich sicher mit Ihrer VPC verbinden, um auf Ihre privaten Endgeräte zuzugreifen.

Nach Abschluss der Tests wird ein Testbericht generiert, der allgemeine Ergebnisse, Low-Level-Logs, Screenshots und Ihre Testartefakte enthält.

Device Farm unterstützt das Testen von nativen und hybriden Android- und iOS-Apps. Weitere Informationen zu unterstützten Testtypen finden Sie unter Test-Frameworks und integrierte Tests in AWS Device Farm.

Terminologie

Device Farm führt die folgenden Begriffe ein, die die Art und Weise definieren, wie Informationen organisiert werden:

Gerätepools

> Eine Sammlung von Geräten, die normalerweise ähnliche Merkmale aufweisen, z. B. Plattform, Hersteller oder Modell.

Auftrag

> Eine Anfrage an Device Farm, eine einzelne App auf einem einzelnen Gerät zu testen. Ein Auftrag umfasst eine oder mehrere Sammlungen.

Gebührenerfassung

> Bezieht sich auf die Fakturierung für Geräte. Sie können Verweise auf zählerüberwachte Geräte oder nicht zählerüberwachte Geräte in der Dokumentation und der API-Referenz finden. Weitere Informationen zur Preisgestaltung finden Sie unter Preise für AWS Device Farm.

project

Ein logischer WorkSpace, der Ausführungen enthält, eine Ausführung pro Test einer einzelnen App auf einem oder mehreren Geräten. Mit Projekten können Sie Workspaces auf Ihre bevorzugte Art und Weise organisieren. Beispielsweise können Sie über ein Projekt pro App-Titel oder ein Projekt pro Plattform verfügen. Sie können so viele Projekte erstellen, wie Sie benötigen.

report

Enthält Informationen zu einem Lauf, bei dem es sich um eine Anforderung an Device Farm handelt, eine einzelne App auf einem oder mehreren Geräten zu testen. Weitere Informationen finden Sie unter Berichte in AWS Device Farm.

run

Ein spezifischer Build Ihrer App mit einer spezifischen Reihe an Tests, die auf einem spezifischen Satz an Geräten ausgeführt werden können. Ein Ausführung erstellt einen Bericht über die Ergebnisse. Eine Ausführung umfasst einen oder mehrere Aufträge. Weitere Informationen finden Sie unter Ausführungen.

Sitzung

Eine Echtzeit-Interaktion mit einem wirklichen physischen Gerät, das über Ihren Webbrowser gehostet wird. Weitere Informationen finden Sie unter Sitzungen.

Suite

Die hierarchische Organisation von Tests in einem Testpaket. Eine Sammlung umfasst einen oder mehrere Tests.

Test

Ein einzelner Testfall in einem Testpaket.

Weitere Informationen über eine Device Farm finden Sie unter Konzepte.

Einrichtung

Informationen zur Verwendung von Device Farm finden Sie unterEinrichtung.

Einrichtung von AWS Device Farm

Bevor Sie Device Farm zum ersten Mal verwenden, müssen Sie die folgenden Aufgaben ausführen:

Themen

Schritt 1: Melden Sie sich an für AWS

Melden Sie sich für Amazon Web Services an (AWS).

Wenn Sie noch keine haben AWS-Konto, führen Sie die folgenden Schritte aus, um eine zu erstellen.

Um sich für eine anzumelden AWS-Konto

1. Öffnen Sie https://portal.aws.amazon.com/billing/die Anmeldung.
2. Folgen Sie den Online-Anweisungen.

   Während der Anmeldung erhalten Sie einen Telefonanruf oder eine Textnachricht und müssen einen Verifizierungscode über die Telefontasten eingeben.

   Wenn Sie sich für eine anmelden AWS-Konto, Root-Benutzer des AWS-Kontoswird eine erstellt. Der Root-Benutzer hat Zugriff auf alle AWS-Services und Ressourcen des Kontos. Als bewährte Sicherheitsmethode weisen Sie einem Benutzer Administratorzugriff zu und verwenden Sie nur den Root-Benutzer, um Aufgaben auszuführen, die Root-Benutzerzugriff erfordern.

Schritt 2: Erstellen oder verwenden Sie einen IAM-Benutzer in Ihrem Konto AWS

Wir empfehlen, dass Sie Ihr AWS Root-Konto nicht für den Zugriff auf Device Farm verwenden. Erstellen Sie stattdessen einen AWS Identity and Access Management (IAM-) Benutzer (oder verwenden Sie einen vorhandenen) in Ihrem AWS Konto und greifen Sie dann mit diesem IAM-Benutzer auf Device Farm zu.

Weitere Informationen finden Sie unter Einen IAM-Benutzer erstellen ().AWS-Managementkonsole

Schritt 3: Erteilen Sie dem IAM-Benutzer die Erlaubnis, auf Device Farm zuzugreifen

Erteilen Sie dem IAM-Benutzer die Erlaubnis, auf Device Farm zuzugreifen. Erstellen Sie dazu eine Zugriffsrichtlinie in IAM und weisen Sie die Zugriffsrichtlinie dann dem IAM-Benutzer wie folgt zu.

> **Note**
>
> Das AWS Root-Konto oder der IAM-Benutzer, mit dem Sie die folgenden Schritte ausführen, muss berechtigt sein, die folgende IAM-Richtlinie zu erstellen und sie an den IAM-Benutzer anzuhängen. Weitere Informationen finden Sie unter Arbeiten mit Richtlinien.

1. Erstellen Sie eine Richtlinie mit dem folgenden JSON-Text. Geben Sie ihm einen aussagekräftigen Titel, z. B. *DeviceFarmAdmin*

   Weitere Informationen zum Erstellen von IAM-Richtlinien finden Sie unter Erstellen von IAM-Richtlinien im IAM-Benutzerhandbuch.
2. Hängen Sie die von Ihnen erstellte IAM-Richtlinie an Ihren neuen Benutzer an. Weitere Informationen zum Anhängen von IAM-Richtlinien an Benutzer finden Sie unter Hinzufügen und Entfernen von IAM-Richtlinien im IAM-Benutzerhandbuch.

Durch das Anhängen der Richtlinie erhält der IAM-Benutzer Zugriff auf alle Device Farm Farm-Aktionen und Ressourcen, die diesem IAM-Benutzer zugeordnet sind. Informationen dazu, wie Sie IAM-Benutzer auf eine begrenzte Anzahl von Gerätefarmaktionen und -ressourcen beschränken können, finden Sie unterIdentitäts- und Zugriffsmanagement in AWS Device Farm.

Nächster Schritt

Sie sind jetzt bereit, Device Farm zu verwenden. Siehe Erste Schritte mit Device Farm.

Erste Schritte mit Device Farm

Diese exemplarische Vorgehensweise zeigt Ihnen, wie Sie Device Farm verwenden, um eine native Android- oder iOS-App zu testen. Sie verwenden die Device Farm Farm-Konsole, um ein Projekt zu erstellen, eine APK- oder IPA-Datei hochzuladen, eine Reihe von Standardtests auszuführen und sich dann die Ergebnisse anzusehen.

> **Note**
>
> Device Farm ist nur in der AWS Region `us-west-2` (Oregon) verfügbar.

Themen

Voraussetzungen

Überprüfen Sie zu Beginn, ob Sie die folgenden Voraussetzungen erfüllt haben:

- Führen Sie die Schritte unter Einrichtung aus. Sie benötigen ein AWS Konto und einen AWS Identity and Access Management (IAM-) Benutzer mit der Berechtigung, auf Device Farm zuzugreifen.

- Für Android können Sie eine APK-Datei (Android-App-Paket) mitbringen oder die von uns bereitgestellte Beispielanwendung verwenden. Für iOS benötigen Sie eine .ipa (iOS App Archive)-Datei. Sie laden die Datei später in dieser exemplarischen Vorgehensweise auf Device Farm hoch.

> **Note**
>
> Stellen Sie sicher, dass Ihre IPA-Datei für ein iOS-Gerät und nicht für einen Simulator erstellt wurde.

- (Optional) Sie benötigen einen Test aus einem der Test-Frameworks, die Device Farm unterstützt. Sie laden dieses Testpaket auf Device Farm hoch und führen den Test dann später in dieser exemplarischen Vorgehensweise aus. Wenn kein Testpaket verfügbar ist, können Sie eine integrierte Standardtestsuite angeben und ausführen. Weitere Informationen finden Sie unter Test-Frameworks und integrierte Tests in AWS Device Farm.

Schritt 1: Melden Sie sich bei der -Konsole an

Sie können die Device Farm Farm-Konsole verwenden, um Projekte und Testläufe zu erstellen und zu verwalten. Informationen zu Projekten und Ausführungen, oder Testläufen, erhalten Sie später in dieser Anleitung.

- Melden Sie sich bei der Device Farm Farm-Konsole unter https://console.aws.amazon.com/devicefarm an.

Schritt 2: Erstellen Sie ein Projekt

Um eine App in Device Farm zu testen, müssen Sie zuerst ein Projekt erstellen.

1. Wählen Sie im Navigationsbereich Mobile Device Testing und dann Projects aus.
2. Wählen Sie unter Projekte zum Testen von Mobilgeräten die Option Projekt erstellen aus.
3. Geben Sie unter Projekt erstellen einen Projektnamen ein (z. B.**`MyDemoProject`**).
4. Wählen Sie Erstellen aus.

   Die Konsole öffnet die Seite Automatisierte Tests Ihres neu erstellten Projekts.

Schritt 3: Einen Lauf erstellen und starten

Nachdem Sie ein Projekt erstellt haben, können Sie jetzt eine Ausführung erstellen und starten. Weitere Informationen finden Sie unter Ausführungen.

1. Wählen Sie auf der Registerkarte Automatisierte Tests die Option Ausführung erstellen aus. Alternativ können Sie dem Tutorial in der Konsole folgen, indem Sie Ausführung mit Tutorial erstellen auswählen.
2. (Optional) Geben Sie unter Run-Einstellungen im Abschnitt Runname einen Namen für Ihren Run ein. Wenn kein Name angegeben wird, benennt die Device Farm-Konsole Ihren Lauf standardmäßig „My Device Farm run“.
3. Wählen Sie unter Laufeinstellungen im Abschnitt Ausführungstyp Ihren Ausführungstyp aus. Wählen Sie Android-App aus, wenn Sie noch keine App zum Testen bereit haben oder wenn Sie eine Android-App (.apk) testen. Wählen Sie iOS-App, wenn Sie eine iOS-App (.ipa) testen.
4. Wählen Sie unter App auswählen im Abschnitt App-Auswahloptionen die Option Von Device Farm bereitgestellte Beispiel-App auswählen aus, wenn Sie keine App zum Testen zur Verfügung haben. Wenn Sie Ihre eigene App mitbringen, wählen Sie Eigene App hochladen und wählen Sie Ihre Anwendungsdatei aus. Wenn Sie eine iOS-App hochladen, müssen Sie darauf achten, iOS device (iOS-Gerät) auszuwählen, und keinen Simulator.
5. Wählen Sie unter Test konfigurieren im Abschnitt Testframework auswählen eines der Testframeworks oder integrierten Testsuiten aus. Informationen zu den jeweiligen Optionen finden Sie unter Test-Frameworks und integrierte Tests.
   - Wenn Sie Ihre Tests für Device Farm noch nicht gepackt haben, wählen Sie Built-In: Fuzz, um eine standardmäßige, integrierte Testsuite auszuführen. Sie können die Standardwerte für Event Count, Event Throttle und Randomizer Seed beibehalten. Weitere Informationen finden Sie unter the section called “Eingebaut: Fuzz (Android und iOS)”.
   - Wenn Sie über ein Testpaket aus einem der unterstützten Test-Frameworks verfügen, wählen Sie das entsprechende Test-Framework aus und laden Sie dann die Datei hoch, die Ihre Tests enthält.
6. Wählen Sie unter Geräte auswählen die Optionen Gerätepool verwenden und Top-Geräte aus.
7. (Optional) Um eine zusätzliche Konfiguration hinzuzufügen, öffnen Sie das Drop-down-Menü Zusätzliche Konfiguration. In diesem Abschnitt können Sie einen der folgenden Schritte ausführen:

   - Um weitere Daten bereitzustellen, die Device Farm während der Ausführung verwenden kann, wählen Sie neben Zusätzliche Daten hinzufügen die Option Datei auswählen aus, und suchen Sie dann die ZIP-Datei, die die Daten enthält, und wählen Sie sie aus.
   - Um eine zusätzliche App für Device Farm zu installieren, die während der Ausführung verwendet werden soll, wählen Sie neben Andere Apps installieren die Option Datei auswählen aus. Suchen Sie dann nach der APK- oder IPA-Datei, die die App enthält, und

wählen Sie sie aus. Wiederholen Sie diesen Vorgang für alle anderen Apps, die Sie installieren möchten. Sie können die Installationsreihenfolge der Apps per Drag & Drop ändern, nachdem Sie diese hochgeladen haben.

- Um anzugeben, ob bei der Ausführung Wi-Fi, Bluetooth, GPS oder NFC aktiviert sein wird, markieren Sie neben Set radio states (Funkstati einstellen) die jeweiligen Felder.
- Geben Sie zur Voreinstellung von Gerätebreite und -länge für den Testlauf neben Device location (Gerätestandort) die Koordinaten ein.
- Um das Geräte-Gebietsschema für die Ausführung voreinzustellen, wählen Sie unter Gerätegebietsschema das Gebietsschema aus.
- Wählen Sie Videoaufnahme aktivieren aus, um während des Tests Videos aufzunehmen.
- Wählen Sie Erfassung von App-Leistungsdaten aktivieren aus, um Leistungsdaten vom Gerät zu erfassen.

> **Note**
>
> Das Einstellen des Funkstatus und des Gebietsschemas des Geräts sind derzeit nur für native Android-Tests verfügbar.

> **Note**
>
> Wenn Sie private Geräte haben, wird auch die für private Geräte spezifische Konfiguration angezeigt.

8. Wählen Sie unten auf der Seite „Lauf erstellen“ aus, um den Lauf zu planen.

Device Farm startet den Lauf, sobald Geräte verfügbar sind, normalerweise innerhalb weniger Minuten. Um den Ausführungsstatus anzuzeigen, wählen Sie auf der Seite Automatisierte Tests Ihres Projekts den Namen Ihres Laufs aus. Auf der Ausführungsseite unter Geräte beginnt jedes Gerät mit dem Symbol „Ausstehend“

in der Gerätetabelle und wechselt dann zum Symbol

„ 

Wird ausgeführt“, wenn der Test beginnt. Nach Abschluss jedes Tests zeigt die Konsole neben dem

Gerätenamen ein Testergebnissymbol an. Wenn alle Tests abgeschlossen sind, ändert sich das Symbol für ausstehende Tests neben dem Testlauf in ein Testergebnissymbol.

Schritt 4: Sehen Sie sich die Ergebnisse des Testlaufs an

Um die Testergebnisse des Laufs anzuzeigen, wählen Sie auf der Seite Automatisierte Tests Ihres Projekts den Namen Ihres Laufs aus. Eine Übersichtsseite wird angezeigt:

- Die Gesamtanzahl der Tests, nach Ergebnis.
- Liste der Tests mit besonderen Warnungen oder Fehlern.
- Eine Liste von Geräten mit Testergebnissen für jedes Gerät.
- Alle während der Ausführung erfassten Bildschirmfotos, gruppiert nach Gerät.
- Ein Abschnitt zum Herunterladen des Analyseergebnisses.

Weitere Informationen finden Sie unter Testberichte in Device Farm anzeigen.

Nächste Schritte

Weitere Informationen über eine Device Farm finden Sie unter Konzepte.

Kauf eines Geräteslots in Device Farm

Sie können die Device Farm Farm-Konsole AWS Command Line Interface (AWS CLI) oder die Device Farm Farm-API verwenden, um einen Geräteslot zu erwerben.

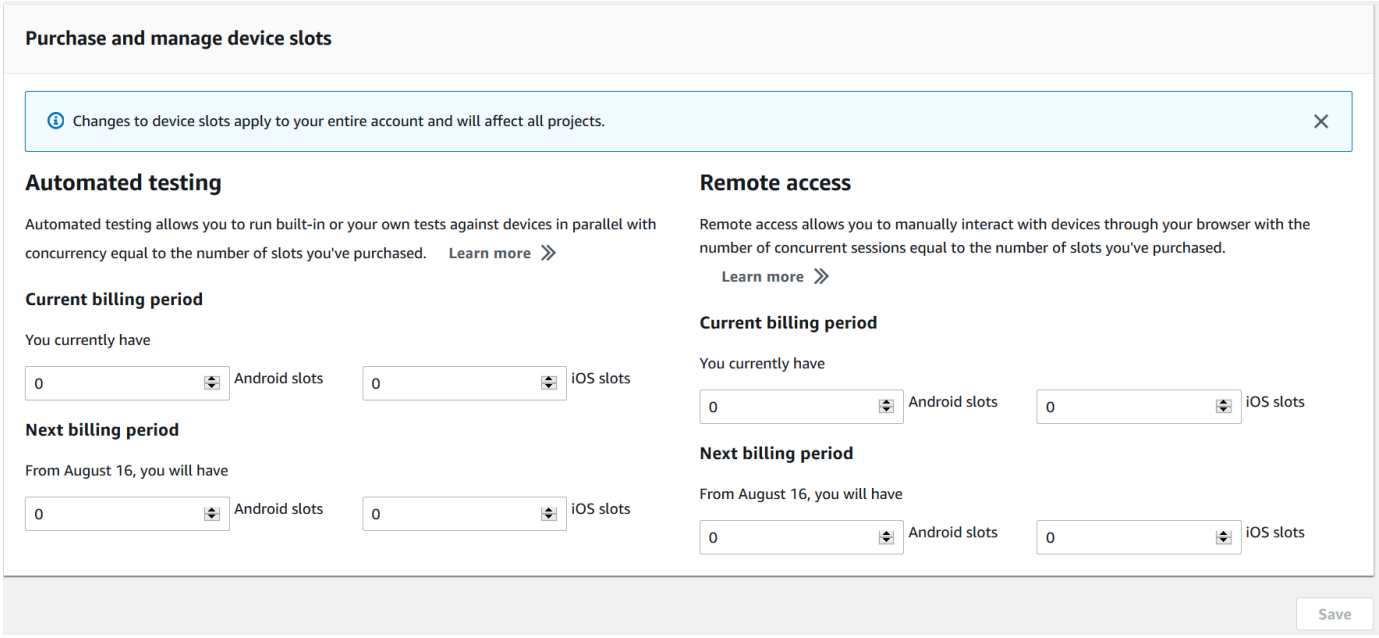
Geräteslots (Konsole) kaufen

1. Melden Sie sich bei der Device Farm Farm-Konsole unter https://console.aws.amazon.com/devicefarm an.
2. Wählen Sie im Navigationsbereich Mobile Device Testing und dann Device slots aus.
3. Auf der Seite Geräteslots kaufen und verwalten können Sie Ihr eigenes Paket erstellen, indem Sie die Anzahl der Steckplätze für Geräte mit automatisiertem Testen und Fernzugriff auswählen, die Sie kaufen möchten. Geben Sie die Anzahl der Steckplätze sowohl für den aktuellen als auch für den nächsten Abrechnungszeitraum an.

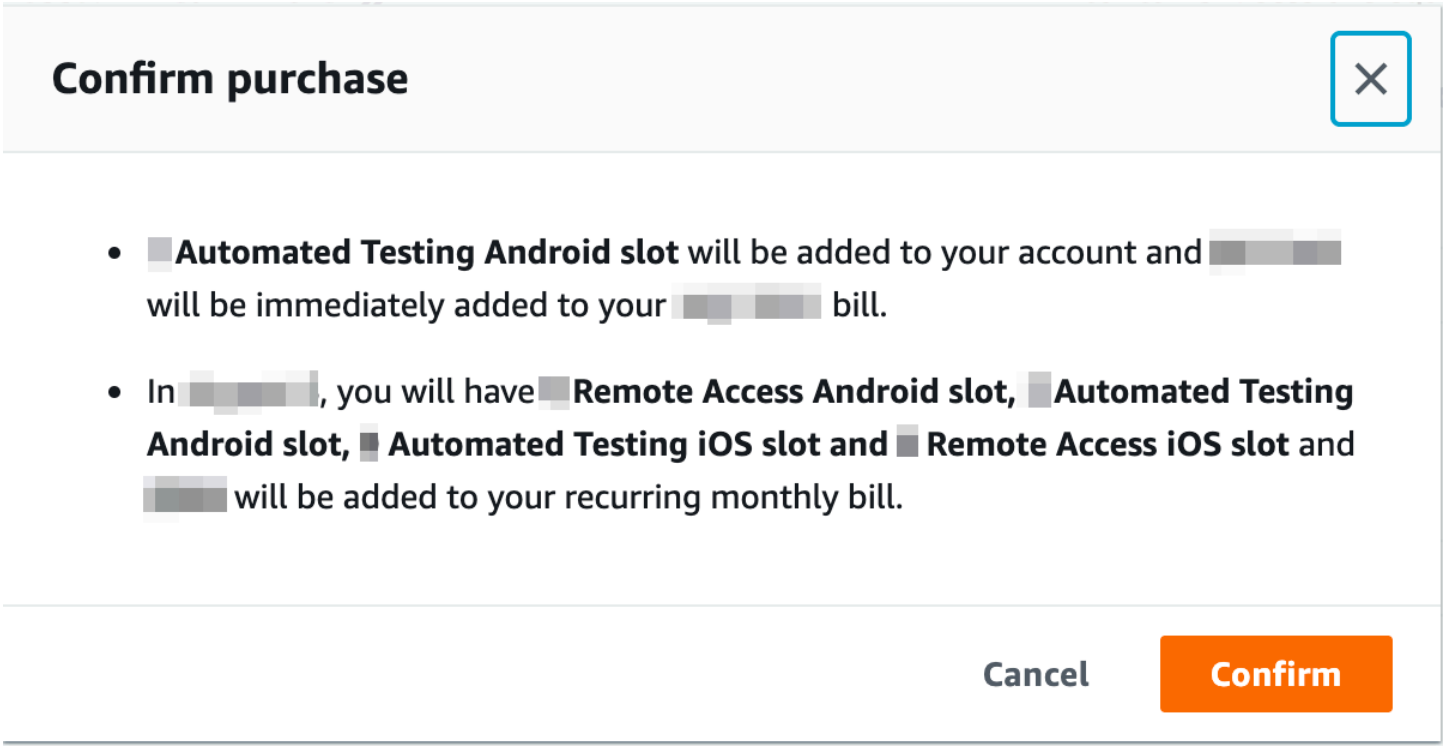
   Wenn Sie die Anzahl der Zeitnischen ändern, wird der Text dynamisch mit dem Rechnungsbetrag aktualisiert. Weitere Informationen finden Sie unter Preise für AWS Device Farm.

   > **Important**
   >
   > Wenn Sie die Anzahl der Geräteslots ändern, aber die Nachricht „Kontaktieren Sie uns“ oder „Kontaktieren Sie uns für einen Kauf“ erhalten, ist Ihr AWS Konto noch nicht für den Kauf der von Ihnen angeforderten Anzahl von Geräteslots zugelassen.
   > Bei diesen Optionen werden Sie aufgefordert, eine E-Mail an das Device Farm Farm-Supportteam zu senden. Geben Sie in der E-Mail die Nummer jedes Gerätetyps an, den Sie kaufen möchten, und für welchen Abrechnungszeitraum.

   > **Note**
   >
   > Änderungen an den Geräteslots gelten für Ihr gesamtes Konto und wirken sich auf alle Projekte aus.



4. Klicken Sie auf Purchase (Kaufen). Das Fenster „Kauf bestätigen“ wird angezeigt. Überprüfen Sie die Informationen und wählen Sie dann Bestätigen, um die Transaktion abzuschließen.



Auf der Seite Geräteslots kaufen und verwalten können Sie sehen, wie viele Geräteslots Sie derzeit haben. Wenn Sie die Anzahl der Plätze erhöht oder verringert haben, sehen Sie die Anzahl der Plätze, über die Sie einen Monat nach dem Datum, an dem Sie die Änderung vorgenommen haben, verfügen werden.

Erwerben Sie einen Geräteslot (AWS CLI)

Sie können den Befehl purchase-offering ausführen, um das Angebot erwerben.

Führen Sie den get-account-settings Befehl aus, um Ihre Device Farm Farm-Kontoeinstellungen aufzulisten, einschließlich der maximalen Anzahl von Geräteslots, die Sie kaufen können, und der Anzahl der verbleibenden kostenlosen Testminuten. Die Ausgabe sieht in etwa wie die folgende aus.

```
{
    "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
        }
    }
}
```

Um die Ihnen zur Verfügung stehenden Geräteplatzangebote aufzulisten, führen Sie den Befehl list-offerings aus. Die Ausgabe sollte folgendermaßen oder ähnlich aussehen:

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

Führen Sie den list-offering-promotions Befehl aus, um die verfügbaren Angebote aufzulisten.

> **Note**
>
> Mit diesem Befehl werden nur Werbeaktionen zurückgegeben, die Sie noch nicht gekauft haben. Sobald Sie einen oder mehrere Plätze über ein Angebot im Rahmen einer Werbeaktion erwerben, erscheint diese Aktion nicht mehr in den Suchergebnissen.

Die Ausgabe sollte folgendermaßen oder ähnlich aussehen:

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

Um den Angebotsstatus zu erhalten, führen Sie den Befehl get-offering-status aus. Die Ausgabe sollte folgendermaßen oder ähnlich aussehen:

```
{
    "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
        }
    }
}
```

Die list-offering-transactions Befehle renew-offering und sind auch für diese Funktion verfügbar. Weitere Informationen hierzu finden Sie unter AWS CLI Referenz.

Erwerben Sie einen Geräteslot (API)

1. Rufen Sie den GetAccountSettingsVorgang auf, um Ihre Kontoeinstellungen aufzulisten.
2. Rufen Sie den ListOfferingsBetrieb auf, um eine Liste der verfügbaren Geräteslots aufzulisten.
3. Rufen Sie den ListOfferingPromotionsBetrieb an, um eine Liste der verfügbaren Angebote und Werbeaktionen zu erhalten.

   > **Note**
   >
   > Dieser Befehl gibt nur Werbeaktionen zurück, die Sie noch nicht gekauft haben. Sobald Sie einen oder mehrere Plätze über eine Werbeaktion erwerben, erscheint diese Aktion nicht mehr in den Suchergebnissen.

4. Rufen Sie den PurchaseOfferingBetrieb auf, um ein Angebot zu erwerben.
5. Rufen Sie den GetOfferingStatusBetrieb auf, um den Status des Angebots abzurufen.

Die ListOfferingTransactionsBefehle RenewOfferingund sind auch für diese Funktion verfügbar.

Hinweise zur Verwendung der Device Farm API finden Sie unterDevice Farm automatisieren.

Einen Geräteslot in Device Farm stornieren

Sie können die Anzahl der Geräteslots sowohl für automatisierte Tests als auch für den Fernzugriff stornieren. Anweisungen finden Sie in einem der folgenden Abschnitte. Der Betrag, der Ihrem Konto für den nächsten Abrechnungszeitraum belastet wurde, wird unter dem Feld Abrechnungszeitraum aufgeführt.

Weitere Informationen zu Geräteslots finden Sie unterKauf eines Geräteslots in Device Farm.

Stornieren Sie einen Geräteslot (Konsole)

1. Melden Sie sich bei der Device Farm Farm-Konsole unter https://console.aws.amazon.com/devicefarm an.
2. Wählen Sie im Navigationsbereich Mobile Device Testing und dann Device slots aus.
3. Auf der Seite Geräteslots kaufen und verwalten können Sie die Anzahl der Geräteslots sowohl für automatisierte Tests als auch für den Fernzugriff verringern, indem Sie den Wert unter Nächster Abrechnungszeitraum verringern. Der Betrag, mit dem Ihr Konto für den nächsten Abrechnungszeitraum belastet wird, wird unter dem Feld Abrechnungszeitraum aufgeführt.
4. Wählen Sie Speichern. Ein Fenster zur Bestätigung der Änderung wird angezeigt. Überprüfen Sie die Informationen und wählen Sie dann Bestätigen, um die Transaktion abzuschließen.

Einen Geräteslot stornieren (AWS CLI)

Sie können den renew-offering Befehl ausführen, um die Anzahl der Geräte für den nächsten Abrechnungszeitraum zu ändern.

Stornieren Sie einen Geräteslot (API)

Rufen Sie den RenewOfferingVorgang auf, um die Anzahl der Geräte in Ihrem Konto zu ändern.

Konzepte von AWS Device Farm

Device Farm ist ein App-Testservice, mit dem Sie Ihre Android-, iOS- und Web-Apps auf echten, physischen Telefonen und Tablets testen und mit ihnen interagieren können, die von Amazon Web Services (AWS) gehostet werden.

In diesem Abschnitt werden wichtige Device Farm Farm-Konzepte beschrieben.

- Geräteunterstützung in AWS Device Farm
- Testumgebungen in AWS Device Farm
- Ausführungen
- Apps
- Berichte in AWS Device Farm
- Sitzungen

Weitere Informationen zu den unterstützten Testtypen in Device Farm finden Sie unterTest-Frameworks und integrierte Tests in AWS Device Farm.

Geräteunterstützung in AWS Device Farm

Die folgenden Abschnitte enthalten Informationen zur Geräteunterstützung in Device Farm.

Themen

- Unterstützte Geräte
- Gerätepools
- Private Geräte
- Branding des Geräts
- Steckplätze für Geräte
- Vorinstallierte Geräte-Apps
- Funktionen der Geräte

Unterstützte Geräte

Device Farm bietet Unterstützung für Hunderte einzigartiger, beliebter Android- und iOS-Geräte und Betriebssystemkombinationen. Die Liste der verfügbaren Geräte wird immer größer, wenn neue Geräte auf den Markt kommen. Die vollständige Liste der Geräte finden Sie in der interaktiven Geräteliste auf Ihrer AWS Konsole.

Gerätepools

Device Farm organisiert seine Geräte in Gerätepools, die Sie für Ihre Tests verwenden können. Diese Gerätepools enthalten verwandte Geräte, z. B. Geräte, die nur auf Android oder nur auf iOS ausgeführt werden. Device Farm bietet kuratierte Gerätepools, z. B. solche für Top-Geräte. Sie können auch Gerätepools mit einer Mischung aus öffentlichen und privaten Geräten erstellen.

Private Geräte

Mit privaten Geräten können Sie genaue Hardware- und Softwarekonfigurationen für Ihre Testanforderungen angeben. Bestimmte Konfigurationen, wie z. B. gerootete Android-Geräte, können als private Geräte unterstützt werden. Jedes private Gerät ist ein physisches Gerät, das Device Farm in Ihrem Namen in einem Amazon-Rechenzentrum bereitstellt. Ihre privaten Geräte sind ausschließlich für Sie für automatische und manuelle Tests verfügbar. Sobald Sie Ihr Abonnement beenden, wird die Hardware aus unserer Umgebung entfernt. Weitere Informationen finden Sie unter Private Geräte und Private Geräte in AWS Device Farm.

Branding des Geräts

Device Farm führt Tests auf physischen Mobil- und Tablet-Geräten von einer Vielzahl von aus OEMs.

Steckplätze für Geräte

Die Geräteplätze werden jeweils parallel verarbeitet: Die Anzahl der Geräteplätze, die Sie erworben haben, legt fest, auf wie viele Geräte in Tests oder Fernzugriffssitzungen parallel zugegriffen werden kann.

Es gibt zwei Typen von Geräteplätzen:

- Ein Geräteplatz für den Remotezugriff ermöglicht, RAS-Sitzungen parallel auszuführen.

Wenn Sie nur einen Geräteplatz für den Remotezugriff haben, können Sie jeweils nur eine Remotezugriffssitzung ausführen. Wenn Sie zusätzliche Geräteplätze für den Remotezugriff erwerben, können Sie mehrere Sitzungen gleichzeitig ausführen.

- Ein Geräteplatz für automatisierte Tests ermöglicht, Tests parallel auszuführen.

  Wenn Sie nur einen Geräteplatz für automatisierte Tests haben, können Sie Ihre Tests jeweils nur auf einem Gerät ausführen. Wenn Sie zusätzliche Geräteplätze für automatisierte Tests erwerben, können Sie Tests gleichzeitig für mehrere Geräten ausführen und erhalten schneller Ergebnisse für mehrere Geräte.

Sie können den Geräteplätze für Gerätefamilien erwerben (Android- oder iOS-Geräte für automatisierte Tests und Android- oder iOS-Geräte für den Remotezugriff). Weitere Informationen finden Sie unter Device Farm – Preise.

Vorinstallierte Geräte-Apps

Geräte in Device Farm enthalten eine kleine Anzahl von Apps, die bereits von Herstellern und Netzbetreibern installiert wurden.

Funktionen der Geräte

Alle Geräte verfügen über eine Internetverbindung. Sie verfügen über keine Transportdienstverbindungen, sodass weder Telefonanrufe getätigt noch SMS-Nachrichten gesendet werden können.

Sie können mit jedem Gerät, das eine Kamera auf der Vorder- oder Rückseite unterstützt, Fotos machen. Abhängig von der Anbringung der Geräte können Fotos dunkel und unscharf wirken.

Google Play Services und Google Chrome sind auf Android-Geräten installiert.

Testumgebungen in AWS Device Farm

AWS Device Farm bietet sowohl benutzerdefinierte als auch standardmäßige Testumgebungen für die Ausführung Ihrer automatisierten Tests. Sie können eine benutzerdefinierte Testumgebung für die vollständige Kontrolle über Ihre automatisierten Tests auswählen. Sie können auch die standardmäßige Standardtestumgebung von Device Farm wählen, die detaillierte Berichte zu jedem Test in Ihrer automatisierten Testsuite bietet.

Themen

Standard-Testumgebung

Wenn Sie einen Test in der Standardumgebung ausführen, bietet Device Farm detaillierte Protokolle und Berichte für jeden Fall in Ihrer Testsuite. Sie können Leistungsdaten, Videos, Screenshots und Protokolle für jeden Test anzeigen, um Probleme Ihrer App zu erkennen und zu beheben.

> **Note**
>
> Da Device Farm detaillierte Berichte in der Standardumgebung bietet, können die Testausführungszeiten länger sein als wenn Sie Ihre Tests lokal ausführen. Wenn Sie schnellere Ausführungszeiten wünschen, führen Sie Ihre Tests in einer benutzerdefinierten Testumgebung aus.

Benutzerdefinierte Testumgebung

Wenn Sie die Testumgebung anpassen, können Sie die Befehle angeben, die Device Farm ausführen soll, um Ihre Tests auszuführen. Dadurch wird sichergestellt, dass Tests auf Device Farm ähnlich wie Tests auf Ihrem lokalen Computer ausgeführt werden. Das Ausführen Ihrer Tests in diesem Modus ermöglicht auch die Live-Protokoll- und Video-Streaming Ihres Tests. Wenn Sie Tests in einer benutzerdefinierten Testumgebung ausführen, erhalten Sie keine präzisen Berichte für jeden Test. Weitere Informationen finden Sie unter Benutzerdefinierte Testumgebungen in AWS Device Farm.

Sie haben die Möglichkeit, eine benutzerdefinierte Testumgebung zu verwenden, wenn Sie die Device Farm Farm-Konsole oder die Device Farm Farm-API verwenden AWS CLI, um einen Testlauf zu erstellen.

Weitere Informationen finden Sie unter Hochladen einer benutzerdefinierten Testspezifikation mithilfe von und. AWS CLIEinen Testlauf in Device Farm erstellen

Läuft in der AWS-Gerätefarm

Die folgenden Abschnitte enthalten Informationen zu Ausführungen in Device Farm.

Eine Ausführung in Device Farm stellt einen bestimmten Build Ihrer App mit einer bestimmten Reihe von Tests dar, die auf einer bestimmten Gruppe von Geräten ausgeführt werden sollen. Bei einem Lauf wird ein Bericht erstellt, der Informationen über die Ergebnisse des Laufs enthält. Eine Ausführung umfasst einen oder mehrere Aufträge.

Themen

Konfiguration ausführen

Im Rahmen eines Laufs können Sie Einstellungen angeben, die Device Farm verwenden kann, um aktuelle Geräteeinstellungen zu überschreiben. Dazu gehören Breiten- und Längengradkoordinaten, zusätzliche Daten (in einer ZIP-Datei enthalten) und Hilfs-Apps (Apps, die vor der zu testenden App installiert werden sollten). Unter Android können einige zusätzliche Einstellungen geändert werden, z. B. das Gebietsschema und der Funkstatus (Bluetooth, GPS, NFC und Wi-Fi).

Führen Sie die Aufbewahrung von Dateien aus

Device Farm speichert Ihre Apps und Dateien 30 Tage lang und löscht sie dann aus dem System. Sie können die Dateien jedoch auch jederzeit selbst löschen.

Device Farm speichert Ihre Laufergebnisse, Protokolle und Screenshots 400 Tage lang und löscht sie dann aus dem System.

Status des Geräts ausführen

Device Farm startet ein Gerät immer neu, bevor es für den nächsten Job verfügbar gemacht wird.

Parallele Läufe

Device Farm führt Tests parallel durch, sobald Geräte verfügbar werden.

Einstellung des Ausführungs-Timeouts

Sie können über einen Wert festlegen, wie lange ein Testlauf ausgeführt werden soll, bevor Sie den Testlauf auf den Geräten stoppen. Beispiel: Wenn Ihr Test für ein Gerät 20 Minuten benötigt, wählen Sie für die Zeitüberschreitung einen Wert von 30 Minuten pro Gerät.

Weitere Informationen finden Sie unter Einstellung des Ausführungszeitlimits für Testläufe in AWS Device Farm.

Werbung in Läufen

Wir empfehlen, dass Sie Anzeigen aus Ihren Apps entfernen, bevor Sie sie auf Device Farm hochladen. Wir können nicht garantieren, dass Werbeanzeigen während der Ausführung angezeigt werden.

Medien in Runs

Sie können Medien oder andere Daten zu Ihrer App bereitstellen. Zusätzliche Daten müssen in einer ZIP-Datei mit einer Größe von höchstens 4 GB bereitgestellt werden.

Allgemeine Aufgaben für Läufe

Weitere Informationen erhalten Sie unter Einen Testlauf in Device Farm erstellen und Testläufe in AWS Device Farm.

Apps in der AWS-Gerätefarm

Die folgenden Abschnitte enthalten Informationen zum Verhalten von Apps in Device Farm.

Themen

- Instrumentierung von Apps
- Apps in Läufen erneut signieren
- Verschleierte Apps in Läufen

Instrumentierung von Apps

Sie müssen Ihre Apps nicht instrumentieren oder Device Farm den Quellcode für Ihre Apps zur Verfügung stellen. Android-Apps können unverändert eingereicht werden. iOS-Apps müssen mit dem Ziel iOS Device (iOS-Gerät) anstelle des Simulators erstellt werden.

Apps in Läufen erneut signieren

Für iOS-Apps müssen Sie Ihrem Provisioning-Profil keine Device Farm UUIDs hinzufügen. Device Farm ersetzt das eingebettete Bereitstellungsprofil durch ein Platzhalterprofil und signiert die App anschließend erneut. Wenn Sie Zusatzdaten bereitstellen, fügt Device Farm sie dem Paket der App hinzu, bevor Device Farm sie installiert, sodass die Hilfsdaten in der Sandbox Ihrer App vorhanden sind. Durch erneutes Signieren der App werden Berechtigungen wie App Group, Associated Domains, Game Center,, HealthKit, Konfiguration von drahtlosem Zubehör HomeKit, In-App-Kauf, Inter-App-Audio, Apple Pay, Push-Benachrichtigungen und VPN-Konfiguration und -Steuerung entfernt.

Bei Android-Apps signiert Device Farm die App erneut. Dadurch könnten alle Funktionen beeinträchtigt werden, die von der Signatur der App abhängen, wie z. B. die Google Maps Android API, oder es könnte die Erkennung von Piraterie- oder Manipulationsschutzmaßnahmen durch Produkte wie auslösen. DexGuard

Verschleierte Apps in Läufen

Wenn die App für Android-Apps verschleiert ist, können Sie sie trotzdem mit Device Farm testen, wenn Sie sie verwenden. ProGuard Wenn Sie die App jedoch DexGuard zusammen mit Anti-Piraterie-Maßnahmen verwenden, kann Device Farm die App nicht erneut signieren und Tests durchführen.

Berichte in AWS Device Farm

Die folgenden Abschnitte enthalten Informationen zu Device Farm Farm-Testberichten.

Themen

- Allgemeine Aufgaben für Berichte

Aufbewahrung von Berichten

Device Farm speichert Ihre Berichte 400 Tage lang. Diese Berichte umfassen Metadaten, Protokolle, Screenshots und Leistungsdaten.

Komponenten des Berichts

Berichte in Device Farm enthalten Pass-and-Fail-Informationen, Absturzberichte, Test- und Geräteprotokolle, Screenshots und Leistungsdaten.

Berichte enthalten sowohl detaillierte Daten zu jedem Gerät sowie allgemeine Ergebnisse wie die Häufigkeit eines bestimmten Problems.

Loggt in Berichten ein

Berichte umfassen vollständige Logcat-Erfassungen für Android-Tests und vollständige Geräte-Konsole-Protokolle für iOS-Tests.

Allgemeine Aufgaben für Berichte

Weitere Informationen finden Sie unter Testberichte in Device Farm anzeigen.

Sitzungen in der AWS-Gerätefarm

Sie können Device Farm verwenden, um interaktive Tests von Android- und iOS-Apps über Fernzugriffssitzungen durchzuführen. Dies umfasst sowohl die manuelle Interaktion in einem Webbrowser als auch das Ausführen von Appium-Tests von einem lokalen Client aus auf dem Remote-Gerät. Entwickler können Probleme mit ihrer App oder ihren Appium-Tests auf einem bestimmten Gerät reproduzieren, um Probleme zu isolieren und zu lösen.

Themen

- Unterstützte Geräte für den Fernzugriff
- Aufbewahrung von Sitzungsdateien
- Instrumentierung von Apps
- Apps in Sitzungen erneut signieren

- Verschleierte Apps in Sitzungen

Unterstützte Geräte für den Fernzugriff

Device Farm bietet Unterstützung für eine Reihe einzigartiger, beliebter Android- und iOS-Geräte. Die Liste der verfügbaren Geräte wird immer größer, wenn neue Geräte auf den Markt kommen. Die Device Farm Farm-Konsole zeigt die aktuelle Liste der Android- und iOS-Geräte an, die für den Fernzugriff verfügbar sind. Weitere Informationen finden Sie unter Geräteunterstützung in AWS Device Farm.

Aufbewahrung von Sitzungsdateien

Device Farm speichert Ihre Apps und Dateien 30 Tage lang und löscht sie dann aus dem System. Sie können die Dateien jedoch auch jederzeit selbst löschen.

Device Farm speichert Ihre Sitzungsprotokolle und aufgenommenen Videos 400 Tage lang und löscht sie dann aus dem System.

Instrumentierung von Apps

Sie müssen Ihre Apps nicht instrumentieren oder Device Farm den Quellcode für Ihre Apps zur Verfügung stellen. Android- und iOS-Apps können unverändert eingereicht werden.

Apps in Sitzungen erneut signieren

Device Farm signiert Android- und iOS-Apps erneut. Dadurch wird möglicherweise Funktionalität beschädigt, die von der Signatur der App abhängt. Beispielsweise hängt die Google Maps-Android-API von der Signatur der App ab. Das erneute Signieren von Apps kann auch die Erkennung von Piraterie oder Manipulation bei Produkten auslösen, z. B. bei Android-Geräten. DexGuard

Verschleierte Apps in Sitzungen

Wenn die App für Android-Apps verschleiert ist, können Sie sie trotzdem mit Device Farm testen, wenn Sie sie verwenden. ProGuard Wenn Sie die App jedoch DexGuard zusammen mit Anti-Piraterie-Maßnahmen verwenden, kann Device Farm die App nicht erneut signieren.

Projekte in AWS Device Farm

Ein Projekt in Device Farm stellt einen logischen Arbeitsbereich in Device Farm dar, der Läufe enthält, einen Lauf für jeden Test einer einzelnen App auf einem oder mehreren Geräten. Projekte ermöglichen es Ihnen, Arbeitsbereiche nach Ihren Wünschen zu organisieren. Beispielsweise kann es ein Projekt pro App-Titel oder ein Projekt pro Plattform geben. Sie können so viele Projekte erstellen, wie Sie benötigen.

Sie können die AWS Device Farm Farm-Konsole AWS Command Line Interface (AWS CLI) oder die AWS Device Farm Farm-API verwenden, um mit Projekten zu arbeiten.

Themen

Ein Projekt in AWS Device Farm erstellen

Sie können ein Projekt mithilfe der AWS Device Farm Farm-Konsole oder der AWS Device Farm Farm-API erstellen. AWS CLI

Voraussetzungen

- Führen Sie die Schritte unter Einrichtung aus.

Erstellen Sie ein Projekt (Konsole)

1. Melden Sie sich bei der Device Farm Farm-Konsole unter https://console.aws.amazon.com/devicefarm an.
2. Wählen Sie im Navigationsbereich Device Farm die Option Mobile Device Testing und dann Projects aus.
3. Wählen Sie New Project (Neues Projekt).
4. Geben Sie einen Namen für Ihr Projekt ein. Optional können Sie einen oder mehrere der folgenden Parameter angeben und dann „Senden“ wählen.

Einstellungen für Virtual Private Cloud (VPC)

Wählen Sie eine VPC, Subnetze und Sicherheitsgruppe aus, die auf das zu testende Gerät und den zugehörigen Testhost angewendet werden sollen. Diese Funktion wird nur mit privaten Geräten unterstützt. Weitere Informationen finden Sie unter VPC-ENI in der AWS-Gerätefarm.

Ausführungsrolle ARN

Eine IAM-Rolle, die der Testrunner in benutzerdefinierten Testumgebungen übernehmen muss. Weitere Informationen finden Sie unter Greifen Sie mithilfe einer IAM-Ausführungsrolle auf AWS-Ressourcen zu.

Umgebungsvariablen

Eine oder mehrere Variablen, die in die Umgebung des Testausführungsrunner-Prozesses eingefügt werden sollen. Variablennamen, die mit „DEVICEFARM_“ beginnen, sind für die Verwendung durch Dienste reserviert. Wir empfehlen, keine sensiblen Werte in diesen Umgebungsvariablen zu speichern und stattdessen eine IAM-Ausführungsrolle zu verwenden, um solche Werte während Ihres Tests von AWS Secrets Manager abzurufen.

Erstellen Sie ein Projekt ()AWS CLI

- Führen Sie create-project unter Angabe eines Projektnamens aus.

  Beispiel:

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

  Die AWS CLI Antwort enthält den Amazon-Ressourcennamen (ARN) des Projekts.

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

Weitere Informationen erhalten Sie unter create-project und AWS CLI Referenz.

Erstellen Sie ein Projekt (API)

- Rufen Sie die `CreateProject`-API auf.

Hinweise zur Verwendung der Device Farm API finden Sie unterDevice Farm automatisieren.

Projektliste in AWS Device Farm anzeigen

Sie können die AWS Device Farm Farm-Konsole oder die AWS Device Farm Farm-API verwenden, um die Liste der Projekte anzuzeigen. AWS CLI

Themen

Voraussetzungen

- Erstellen Sie mindestens ein Projekt in Device Farm. Befolgen Sie die Anweisungen unter Ein Projekt in AWS Device Farm erstellen, und kehren Sie dann zu dieser Seite zurück.

Sehen Sie sich die Projektliste an (Konsole)

1. Melden Sie sich bei der Device Farm Farm-Konsole unter https://console.aws.amazon.com/devicefarm an.
2. Gehen Sie wie folgt vor, um die Liste der verfügbaren Projekte zu finden:
   - Wählen Sie für Testprojekte für mobile Geräte im Navigationsmenü Device Farm die Option Testen mobiler Geräte und dann Projekte aus.
   - Wählen Sie für Desktop-Browser-Testprojekte im Navigationsmenü Device Farm die Option Desktop Browser Testing und dann Projekte aus.

Sehen Sie sich die Projektliste an (AWS CLI)

- Um die Projektliste anzuzeigen, führen Sie den Befehl list-projects aus.

  Um Informationen zu einem einzigen Projekt anzuzeigen, führen Sie den Befehl get-project aus.

Informationen zur Verwendung von Device Farm mit dem AWS CLI finden Sie unterAWS CLI Referenz.

Sehen Sie sich die Projektliste an (API)

- Um die Projektliste anzuzeigen, rufen Sie die `ListProjects`-API auf.

  Um Informationen zu einem einzigen Projekt anzuzeigen, rufen Sie die `GetProject`-API auf.

Informationen zur AWS Device Farm Farm-API finden Sie unterDevice Farm automatisieren.

Testläufe in AWS Device Farm

Eine Ausführung in Device Farm stellt einen bestimmten Build Ihrer App mit einer bestimmten Reihe von Tests dar, die auf einer bestimmten Gruppe von Geräten ausgeführt werden sollen. Bei einem Lauf wird ein Bericht erstellt, der Informationen zu den Ergebnissen des Laufs enthält. Eine Ausführung umfasst einen oder mehrere Aufträge. Weitere Informationen finden Sie unter Ausführungen.

Sie können die AWS Device Farm Farm-Konsole AWS Command Line Interface (AWS CLI) oder die AWS Device Farm Farm-API verwenden, um mit Testläufen zu arbeiten.

Themen

Einen Testlauf in Device Farm erstellen

Sie können die Device Farm Farm-Konsole oder die Device Farm Farm-API verwenden AWS CLI, um einen Testlauf zu erstellen. Sie können auch ein unterstütztes Plugin verwenden, z. B. die Jenkins- oder Gradle-Plugins für Device Farm. Weitere Informationen zu den Plugins finden Sie unter Tools und Plugins. Weitere Informationen zu Testläufen finden Sie unter Ausführungen.

Themen

Voraussetzungen

Sie müssen ein Projekt in Device Farm haben. Befolgen Sie die Anweisungen unter Ein Projekt in AWS Device Farm erstellen, und kehren Sie dann zu dieser Seite zurück.

Erstellen Sie einen Testlauf (Konsole)

1. Melden Sie sich bei der Device Farm Farm-Konsole unter https://console.aws.amazon.com/devicefarm an.
2. Wählen Sie im Navigationsbereich Mobile Device Testing und dann Projects aus.
3. Wenn Sie bereits über ein Projekt verfügen, können Sie Ihre Tests in das Projekt hochladen. Wählen Sie andernfalls Neues Projekt aus, geben Sie einen Projektnamen ein und wählen Sie dann Erstellen aus.
4. Öffnen Sie Ihr Projekt und wählen Sie dann Create run aus.
5. (Optional) Geben Sie unter Run-Einstellungen im Abschnitt Runname einen Namen für Ihren Run ein. Wenn kein Name angegeben wird, benennt die Device Farm-Konsole Ihren Lauf standardmäßig „My Device Farm run“.
6. (Optional) Unter Ausführungseinstellungen können Sie im Abschnitt Job-Timeout das Ausführungstimeout für Ihren Testlauf angeben. Wenn Sie eine unbegrenzte Anzahl von Testslots verwenden, vergewissern Sie sich, dass unter Abrechnungsmethode die Option Unmetered ausgewählt ist.
7. Wählen Sie unter Laufeinstellungen im Abschnitt Ausführungstyp Ihren Ausführungstyp aus. Wählen Sie Android-App aus, wenn Sie noch keine App zum Testen bereit haben oder wenn Sie eine Android-App (.apk) testen. Wählen Sie iOS-App, wenn Sie eine iOS-App (.ipa) testen. Wählen Sie Web-App aus, wenn Sie Webanwendungen testen möchten.
8. Wählen Sie unter App auswählen im Abschnitt App-Auswahloptionen die Option Von Device Farm bereitgestellte Beispiel-App auswählen aus, wenn Sie keine App zum Testen zur Verfügung haben. Wenn Sie Ihre eigene App mitbringen, wählen Sie Eigene App hochladen und wählen Sie Ihre Anwendungsdatei aus. Wenn Sie eine iOS-App hochladen, müssen Sie darauf achten, iOS device (iOS-Gerät) auszuwählen, und keinen Simulator.
9. Wählen Sie unter Test konfigurieren eines der verfügbaren Test-Frameworks aus.

> **Note**
>
> Wenn Sie keine Tests verfügbar haben, wählen Sie Built-in: Fuzz (Integriert: Fuzz) zum Starten einer integrierten Standard-Testreihe. Wenn Sie Built-in: Fuzz (Integriert: Fuzz)

wählen und die Felder Event count (Ereignisanzahl), Event throttle (Ereignisdrosselung) und Randomizer seed (Randomisierungs-Seed) angezeigt werden, können Sie die Werte ändern oder beibehalten.

Weitere Informationen zu den verfügbaren Testreihen finden Sie unter Test-Frameworks und integrierte Tests in AWS Device Farm.

10. Wenn Sie Built-In: Fuzz nicht ausgewählt haben, wählen Sie unter Testpaket auswählen die Option Datei auswählen aus. Suchen Sie die Datei, die Ihre Tests enthält, und wählen Sie sie aus.
11. Wählen Sie für Ihre Testumgebung die Option Test in unserer Standardumgebung ausführen oder Test in einer benutzerdefinierten Umgebung ausführen aus. Weitere Informationen finden Sie unter Testumgebungen in AWS Device Farm.
12. Wenn Sie eine benutzerdefinierte Testumgebung verwenden, können Sie optional Folgendes tun:

    - Wenn Sie die Standard-Testspezifikation in einer benutzerdefinierten Testumgebung bearbeiten möchten, wählen Sie Edit (Bearbeiten), um die Standard-YAML-Spezifikation zu aktualisieren.
    - Wenn Sie die Testspezifikation geändert haben, wählen Sie Als neu speichern, um sie zu aktualisieren.
    - Sie können Umgebungsvariablen konfigurieren. Die hier angegebenen Variablen haben Vorrang vor allen Variablen, die im übergeordneten Projekt konfiguriert werden können.
13. Führen Sie unter Geräte auswählen einen der folgenden Schritte aus:

    - Um einen integrierten Gerätepool zu wählen, für den die Tests ausgeführt werden sollen, wählen Sie für Device pool (Gerätepool) die Option Top Devices (Top-Geräte).
    - Um einen eigenen Gerätepool zu erstellen, für den die Tests ausgeführt werden sollen, befolgen Sie die Anweisungen in Einen Gerätepool erstellen und kehren dann zu dieser Seite zurück.
    - Wenn Sie zuvor bereits einen eigenen Gerätepool erstellt haben, wählen Sie für Device pool (Gerätepool) diesen Gerätepool aus.
    - Wählen Sie Geräte manuell auswählen und wählen Sie die gewünschten Geräte aus, für die Sie den Betrieb ausführen möchten. Diese Konfiguration wird nicht gespeichert.

Weitere Informationen finden Sie unter Geräteunterstützung in AWS Device Farm.

14. (Optional) Um eine zusätzliche Konfiguration hinzuzufügen, öffnen Sie das Drop-down-Menü Zusätzliche Konfiguration. In diesem Abschnitt können Sie einen der folgenden Schritte ausführen:

    - Verwenden Sie das ARN-Feld Ausführungsrolle, um einen ARN für die Ausführungsrolle bereitzustellen oder einen im übergeordneten Projekt konfigurierten zu überschreiben.
    - Um weitere Daten bereitzustellen, die Device Farm während der Ausführung verwenden kann, wählen Sie neben Zusätzliche Daten hinzufügen die Option Datei auswählen aus, und suchen Sie dann die ZIP-Datei, die die Daten enthält, und wählen Sie sie aus.
    - Um eine zusätzliche App für Device Farm zu installieren, die während der Ausführung verwendet werden soll, wählen Sie neben Andere Apps installieren die Option Datei auswählen aus. Suchen Sie dann nach der APK- oder IPA-Datei, die die App enthält, und wählen Sie sie aus. Wiederholen Sie diesen Vorgang für alle anderen Apps, die Sie installieren möchten. Sie können die Installationsreihenfolge der Apps per Drag & Drop ändern, nachdem Sie diese hochgeladen haben.
    - Um anzugeben, ob bei der Ausführung Wi-Fi, Bluetooth, GPS oder NFC aktiviert sein wird, markieren Sie neben Set radio states (Funkstati einstellen) die jeweiligen Felder.
    - Geben Sie zur Voreinstellung von Gerätebreite und -länge für den Testlauf neben Device location (Gerätestandort) die Koordinaten ein.
    - Um das Geräte-Gebietsschema für die Ausführung voreinzustellen, wählen Sie unter Gerätegebietsschema das Gebietsschema aus.
    - Wählen Sie Videoaufnahme aktivieren aus, um während des Tests Videos aufzunehmen.
    - Wählen Sie Erfassung von App-Leistungsdaten aktivieren aus, um Leistungsdaten vom Gerät zu erfassen.

    > **Note**
    >
    > Das Einstellen des Funkstatus und des Gebietsschemas des Geräts sind derzeit nur für native Android-Tests verfügbar.

> **Note**
>
> Wenn Sie private Geräte haben, wird auch die für private Geräte spezifische Konfiguration angezeigt.

15. Wählen Sie unten auf der Seite „Lauf erstellen“ aus, um den Lauf zu planen.

Device Farm startet den Lauf, sobald Geräte verfügbar sind, normalerweise innerhalb weniger Minuten. Während Ihres Testlaufs zeigt die Device Farm Farm-Konsole

in der Ausführungstabelle ein ausstehendes Symbol an. Jedes Gerät, das gerade ausgeführt wird, beginnt ebenfalls mit dem Symbol „Ausstehend“ und wechselt dann zu Beginn des Tests zum Symbol „ 

Wird ausgeführt“. Nach Abschluss jedes Tests wird neben dem Gerätenamen ein Testergebnissymbol angezeigt. Wenn alle Tests abgeschlossen sind, ändert sich das Symbol für ausstehende Tests neben dem Testlauf in ein Testergebnissymbol.

Informationen zum Beenden des Testlaufs finden Sie unterStoppen eines Laufs in AWS Device Farm.

Erstellen Sie einen Testlauf (AWS CLI)

Sie können den verwenden AWS CLI , um einen Testlauf zu erstellen.

Themen

- Schritt 1: Wählen Sie ein Projekt
- Schritt 2: Wählen Sie einen Gerätepool
- Schritt 3: Laden Sie Ihre Anwendungsdatei hoch
- Schritt 4: Laden Sie Ihr Testskript-Paket hoch
- Schritt 5: (Optional) Laden Sie Ihre benutzerdefinierte Testspezifikation hoch
- Schritt 6: Einen Testlauf planen

Schritt 1: Wählen Sie ein Projekt

Sie müssen Ihren Testlauf einem Device Farm Farm-Projekt zuordnen.

1. Führen Sie den Befehl aus, um Ihre Device Farm Farm-Projekte aufzulistenlist-projects. Wenn Sie über kein Projekt verfügen, finden Sie weitere Informationen unter Ein Projekt in AWS Device Farm erstellen.

   Beispiel:

   ```
   aws devicefarm list-projects
   ```

   Die Antwort enthält eine Liste Ihrer Device Farm Farm-Projekte.

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

2. Wählen Sie ein Projekt aus, das mit dem Testlauf verknüpft werden soll, und notieren Sie sich den zugehörigen Amazon-Ressourcennamen (ARN).

Schritt 2: Wählen Sie einen Gerätepool

Sie müssen einen Gerätepool auswählen, der mit Ihrem Testlauf verknüpft werden soll.

1. Um Ihre Gerätepools anzuzeigen, führen Sie list-device-pools unter Angabe Ihres Projekt-ARN aus.

   Beispiel:

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

   Die Antwort umfasst die integrierten Gerätefarm-Gerätepools wieTop Devices, und alle Gerätepools, die zuvor für dieses Projekt erstellt wurden:

   ```
   {
       "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. Wählen Sie einen Gerätepool aus und notieren Sie sich den zugehörigen ARN.

   Sie können auch einen Gerätepool erstellen und dann zu diesem Schritt zurückkehren. Weitere Informationen finden Sie unter Erstellen Sie einen Gerätepool (AWS CLI).

Schritt 3: Laden Sie Ihre Anwendungsdatei hoch

Um Ihre Upload-Anfrage zu erstellen und eine vorsignierte Upload-URL für Amazon Simple Storage Service (Amazon S3) zu erhalten, benötigen Sie:

- Ihren Projekt-ARN.
- Name Ihrer App-Datei.

- Typ des Uploads.

Weitere Informationen finden Sie unter create-upload.

1. Führen Sie zum Hochladen einer Datei create-upload mit den Parametern `--project-arn`, `--name` und `--type` aus.

   Das folgende Beispiel erstellt einen Upload für eine Android-App:

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

   Die Antwort enthält Ihren App-Upload-ARN und eine vorsignierte URL.

   ```
   {
       "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. Notieren Sie sich den App-Upload-ARN und die vorsignierte URL.
3. Laden Sie Ihre App-Datei mit der vorsignierten Amazon S3 S3-URL hoch. Dieses Beispiel verwendet curl zum Hochladen einer Android-.apk-Datei:

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

   Weitere Informationen finden Sie unter Hochladen von Objekten mithilfe von Presigned URLs im Amazon Simple Storage Service-Benutzerhandbuch.
4. Um den Status Ihres App-Uploads zu überprüfen, führen Sie get-upload unter Angabe des ARN des App-Uploads aus.

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

Warten Sie, bis der Status in der Antwort SUCCEEDED lautet, bevor Sie Ihr Testskript-Paket hochladen.

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

Schritt 4: Laden Sie Ihr Testskript-Paket hoch

Als Nächstes laden Sie Ihr Testskript-Paket hoch.

1. Um Ihre Upload-Anfrage zu erstellen und eine vorsignierte Amazon S3 S3-Upload-URL zu erhalten, führen Sie den create-upload Befehl mit den `--type` Parametern `--project-arn--name`, und aus.

   Dieses Beispiel erstellt einen Appium Java TestNG-Testpaket-Upload:

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

   Die Antwort enthält Ihren Testpaket-Upload-ARN und eine vorsignierte URL.

   ```
   {
       "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. Notieren Sie sich den ARN des Testpaket-Uploads und die vorsignierte URL.
3. Laden Sie Ihre Testskript-Paketdatei mit der vorsignierten Amazon S3 S3-URL hoch. Dieses Beispiel verwendet curl zum Hochladen einer Appium TestNG-Skript-ZIP-Datei:

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

4. Um den Status Ihres Testskript-Paket-Uploads zu überprüfen, führen Sie get-upload unter Angabe des ARN des Testpaket-Uploads aus Schritt 1 aus.

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

Warten Sie, bis der Status in der Antwort SUCCEEDED lautet, bevor Sie mit dem nächsten, optionalen Schritt fortfahren.

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

Schritt 5: (Optional) Laden Sie Ihre benutzerdefinierte Testspezifikation hoch

Wenn Sie Ihre Tests in einer Standard-Testumgebung ausführen, können Sie diesen Schritt überspringen.

Device Farm verwaltet eine Standardtestspezifikationsdatei für jeden unterstützten Testtyp. Als Nächstes laden Sie Ihre Standard-Testspezifikation herunter und verwenden sie zum Erstellen eines benutzerdefinierten Testspezifikation-Upload für die Ausführung Ihrer Tests in einer benutzerdefinierten Testumgebung. Weitere Informationen finden Sie unter Testumgebungen in AWS Device Farm.

1. Um den Upload-ARN für Ihre Standard-Testspezifikation zu finden, führen Sie list-uploads unter Angabe Ihres Projekt-ARN aus.

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

   Die Antwort enthält einen Eintrag für jede Standard-Testspezifikation:

   ```
   {
       "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. Wählen Sie Ihre Standard-Testspezifikation aus der Liste aus. Notieren Sie deren Upload-ARN.
3. Um Ihre Standard-Spezifikation herunterzuladen, führen Sie get-upload unter Angabe des Upload-ARN aus.

   Beispiel:

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

Die Antwort enthält eine vorsignierte URL zu dem Speicherort, von dem Sie Ihre Standard-Testspezifikation herunterladen können.

4. Dieses Beispiel verwendet curl, um die Standard-Testspezifikation herunterzuladen und als Datei unter dem Namen `MyTestSpec.yml` zu speichern:

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

5. Sie können die Standard-Testspezifikation entsprechend Ihrer Testanforderungen bearbeiten und die abgewandelte Testspezifikation dann in zukünftigen Testläufen verwenden. Überspringen Sie diesen Schritt, um die Standard-Testspezifikation unverändert in einer benutzerdefinierten Testumgebung zu verwenden.

6. Um einen Upload Ihrer benutzerdefinierten Testspezifikation zu erstellen, führen Sie create-upload unter Angabe des Namens und Typs Ihrer Testspezifikation und des Projekt-ARN aus.

   Dieses Beispiel erstellt einen Upload für eine benutzerdefinierte Appium Java TestNG-Testspezifikation:

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

   Die Antwort enthält den Testspezifikations-Upload-ARN und und die vorsignierte URL:

```
{
    "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. Notieren Sie sich den ARN des Testspezifikations-Uploads und die vorsignierte URL.
8. Laden Sie Ihre Testspezifikationsdatei mit der vorsignierten Amazon S3 S3-URL hoch. In diesem Beispiel wird curl eine Appium JavaTest NG-Testspezifikation hochgeladen:

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

9. Um den Status Ihres Testspezifikations-Uploads zu überprüfen, führen Sie get-upload unter Angabe des ARN des Uploads aus.

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

Warten Sie, bis der Status in der Antwort SUCCEEDED lautet, bevor Sie Ihren Testlauf planen.

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

Um Ihre benutzerdefinierte Testspezifikation zu aktualisieren, führen Sie update-upload unter Angabe des Upload-ARN für die Testspezifikation aus. Weitere Informationen finden Sie unter update-upload.

Schritt 6: Einen Testlauf planen

Um einen Testlauf mit dem AWS CLI, run, zu planenschedule-run, geben Sie Folgendes an:

- Projekt-ARN von Schritt 1.
- Gerätepool-ARN von Schritt 2.
- App-Upload-ARN von Schritt 3.

- Testpaket-Upload-ARN von Schritt 4.

Wenn Sie Tests in einer benutzerdefinierten Umgebung ausführen, benötigen Sie außerdem Ihren Testspezifikation-ARN von Schritt 5.

So planen Sie einen Testlauf in einer Standard-Testumgebung

- Führen Sie schedule-run unter Angabe von Projekt-ARN, Gerätepool-ARN, Anwendungs-Upload-ARN und Testpaketinformationen aus.

  Beispiel:

```
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
```

  Die Antwort enthält einen Testlauf-ARN, mittels dem Sie den Status Ihres Testlaufs überprüfen können.

```
{
    "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
        }
    }
}
```

Weitere Informationen finden Sie unter schedule-run.

So planen Sie einen Testlauf in einer benutzerdefinierten Testumgebung

- Die Schritte sind fast identisch mit denen für die Standard-Testumgebung mit einem zusätzlichen `testSpecArn`-Attribut im Parameter `--test`.

  Beispiel:

```
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
```

So überprüfen Sie den Status Ihres Testlaufs

- Verwenden Sie den Befehl get-run und geben Sie den Testlauf-ARN an:

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

Weitere Informationen finden Sie unter get-run. Informationen zur Verwendung von Device Farm mit dem AWS CLI finden Sie unterAWS CLI Referenz.

Erstellen Sie einen Testlauf (API)

Die Schritte sind dieselben wie im AWS CLI Abschnitt beschrieben. Siehe Erstellen Sie einen Testlauf (AWS CLI).

Sie benötigen folgende Informationen zum Aufrufen der `ScheduleRun` API:

- Projekt-ARN. Siehe Erstellen Sie ein Projekt (API) und `CreateProject`.
- App-Upload-ARN. Siehe `CreateUpload`.
- Testpaket Upload-ARN. Siehe `CreateUpload`.
- Gerätepool-ARN. Siehe Einen Gerätepool erstellen und `CreateDevicePool`.

> **Note**
>
> Wenn Sie Tests in einer benutzerdefinierten Testumgebung ausführen, benötigen Sie außerdem Ihren Testspezifikation-Upload-ARN. Weitere Informationen erhalten Sie unter Schritt 5: (Optional) Laden Sie Ihre benutzerdefinierte Testspezifikation hoch und `CreateUpload`.

Hinweise zur Verwendung der Device Farm API finden Sie unterDevice Farm automatisieren.

Nächste Schritte

In der Device Farm Farm-Konsole

ändert sich das Uhrsymbol in ein Ergebnissymbol, wie z. B.

Erfolg 

wenn die Ausführung abgeschlossen ist. Sobald die Tests abgeschlossen sind, wird ein Bericht für den Testlauf angezeigt. Weitere Informationen finden Sie unter Berichte in AWS Device Farm.

Befolgen Sie zur Nutzung des Berichts die Anweisungen unter Testberichte in Device Farm anzeigen.

Einstellung des Ausführungszeitlimits für Testläufe in AWS Device Farm

Sie können über einen Wert festlegen, wie lange ein Testlauf ausgeführt werden soll, bevor Sie den Testlauf auf den Geräten stoppen. Der Standardwert des Ausführungstimeouts ist 150 Minuten pro Gerät, aber Sie können den Wert auf bis zu 5 Minuten heruntersetzen. Sie können die AWS Device Farm Farm-Konsole oder die AWS Device Farm Farm-API verwenden AWS CLI, um das Ausführungstimeout festzulegen.

> **Important**
>
> Die Option des Ausführungstimeouts sollte auf die maximale Dauer für einen Testlauf festgelegt werden, zuzüglich etwas Pufferzeitraum. Beispiel: Wenn Ihr Test für ein Gerät 20 Minuten benötigt, wählen Sie als Timeout einen Wert von 30 Minuten pro Gerät.

Wenn die Ausführung auf einem Gerät den Timeoutwert überschreitet, wird die Ausführung für dieses Gerät zwangsweise beendet. Möglicherweise liefert der Test Teilergebnisse. Wenn Sie die zeitgenaue Abrechnungsoption verwenden, wird die Ausführung bis zu diesem Zeitpunkt in Rechnung gestellt. Weitere Informationen zur Preisgestaltung finden Sie unter Device Farm Pricing.

Sie können diese Funktion verwenden, wenn Sie wissen, wie lange die Ausführung eines Tests auf jedem Gerät dauern sollte. Wenn Sie eine Zeitbeschränkung für die Ausführung eines Test festlegen, können Sie vermeiden, dass bei Testläufen, die aus irgendeinem Grund "hängen", Geräteminuten in Rechnung gestellt werden, in denen keine Tests ausgeführt werden. Mit anderen Worten, Sie können mithilfe der Timeoutfunktion die Ausführung von Tests stoppen, wenn diese länger dauern als erwartet.

Sie können die Zeitbeschränkung für die Ausführung an zwei Stellen einstellen: auf Projektebene und auf Ebene des Testlaufs.

Voraussetzungen

1. Führen Sie die Schritte unter Einrichtung aus.
2. Erstellen Sie ein Projekt in Device Farm. Befolgen Sie die Anweisungen unter Ein Projekt in AWS Device Farm erstellen, und kehren Sie dann zu dieser Seite zurück.

Legen Sie das Ausführungs-Timeout für ein Projekt fest

1. Melden Sie sich bei der Device Farm Farm-Konsole unter https://console.aws.amazon.com/devicefarm an.
2. Wählen Sie im Navigationsbereich Device Farm die Option Mobile Device Testing und dann Projects aus.
3. Wenn Sie bereits ein Projekt haben, wählen Sie es aus der Liste aus. Andernfalls wählen Sie Neues Projekt, geben Sie einen Namen für Ihr Projekt ein und wählen Sie dann Absenden.
4. Wählen Sie Project settings (Projekteinstellungen) aus.
5. Geben Sie auf der Registerkarte General (Allgemein) für Execution Timeout (Ausführungstimeout) einen Wert ein oder verwenden Sie den Schieberegler.
6. Wählen Sie Speichern.

   Alle Testläufe in Ihrem Projekt verwenden nun den Wert für die Zeitbeschränkung der Ausführung, den Sie angegeben haben, es sei denn, Sie überschreiben den Zeitüberschreitungswert, wenn Sie einen Testlauf planen.

Legen Sie das Ausführungs-Timeout für einen Testlauf fest

1. Melden Sie sich bei der Device Farm Farm-Konsole unter https://console.aws.amazon.com/devicefarm an.
2. Wählen Sie im Navigationsbereich Device Farm die Option Mobile Device Testing und dann Projects aus.
3. Wenn Sie bereits ein Projekt haben, wählen Sie es aus der Liste aus. Andernfalls wählen Sie Neues Projekt, geben Sie einen Namen für Ihr Projekt ein und wählen Sie dann Absenden.
4. Wählen Sie Create a new run (Neuen Lauf erstellen).
5. Befolgen Sie die Schritte zum Auswählen einer Anwendung, konfigurieren Ihren Test, wählen Sie die Geräte aus und geben Sie einen Gerätestatus an.
6. Geben Sie unter Überprüfen und Ausführung starten für Ausführungstimeout festlegen einen Wert ein, oder verwenden Sie den Schieberegler.
7. Wählen Sie Confirm and start run (Bestätigen und Testlauf starten).

Simulation von Netzwerkverbindungen und -bedingungen für Ihre AWS Device Farm Farm-Läufe

Sie können Network Shaping verwenden, um Netzwerkverbindungen und -bedingungen zu simulieren, während Sie Ihre Android-, iOS- und Web-Apps in Device Farm testen. Sie können beispielsweise eine verlustbehaftete oder unterbrochene Internetverbindung simulieren.

Wenn Sie eine Ausführung unter Verwendung der Standard-Netzwerkeinstellungen erstellen, verfügt jedes Gerät über eine vollständige, ungehinderte Wi-Fi-Verbindung mit Internet-Konnektivität. Wenn Sie Network Shaping verwenden, können Sie die Wi-Fi-Verbindung ändern, um ein Netzwerkprofil wie 3G oder Lossy anzugeben, das den Durchsatz, WiFi die Verzögerung, den Jitter und den Verlust sowohl für eingehenden als auch für ausgehenden Verkehr steuert.

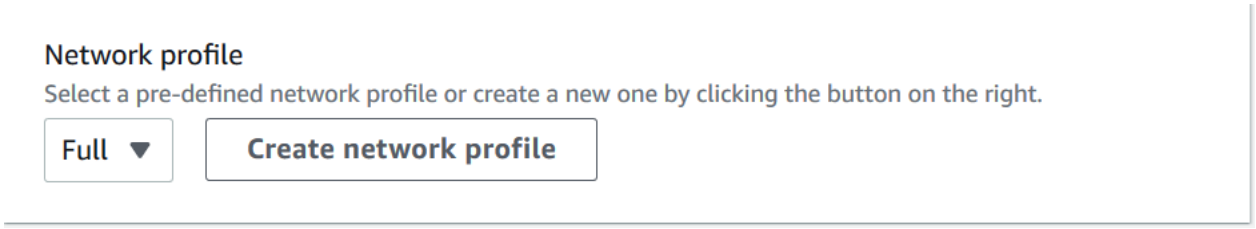
Themen

Richten Sie Network Shaping ein, wenn Sie einen Testlauf planen

Wenn Sie einen Lauf planen, können Sie aus einem der von Device Farm kuratierten Profile wählen oder Ihr eigenes Profil erstellen und verwalten.

1. Wählen Sie in einem beliebigen Device Farm Farm-Projekt Create a new run aus.

   Falls Sie noch keine Projekte besitzen, finden Sie unter Ein Projekt in AWS Device Farm erstellen weitere Informationen.
2. Wählen Sie Ihre Anwendung und dann Weiter aus.
3. Konfigurieren Sie Ihren Test und wählen Sie dann Weiter.
4. Wählen Sie Ihre Geräte aus und wählen Sie dann Weiter.
5. Wählen Sie im Abschnitt Standort- und Netzwerkeinstellungen ein Netzwerkprofil aus, oder wählen Sie Netzwerkprofil erstellen, um Ihr eigenes zu erstellen.

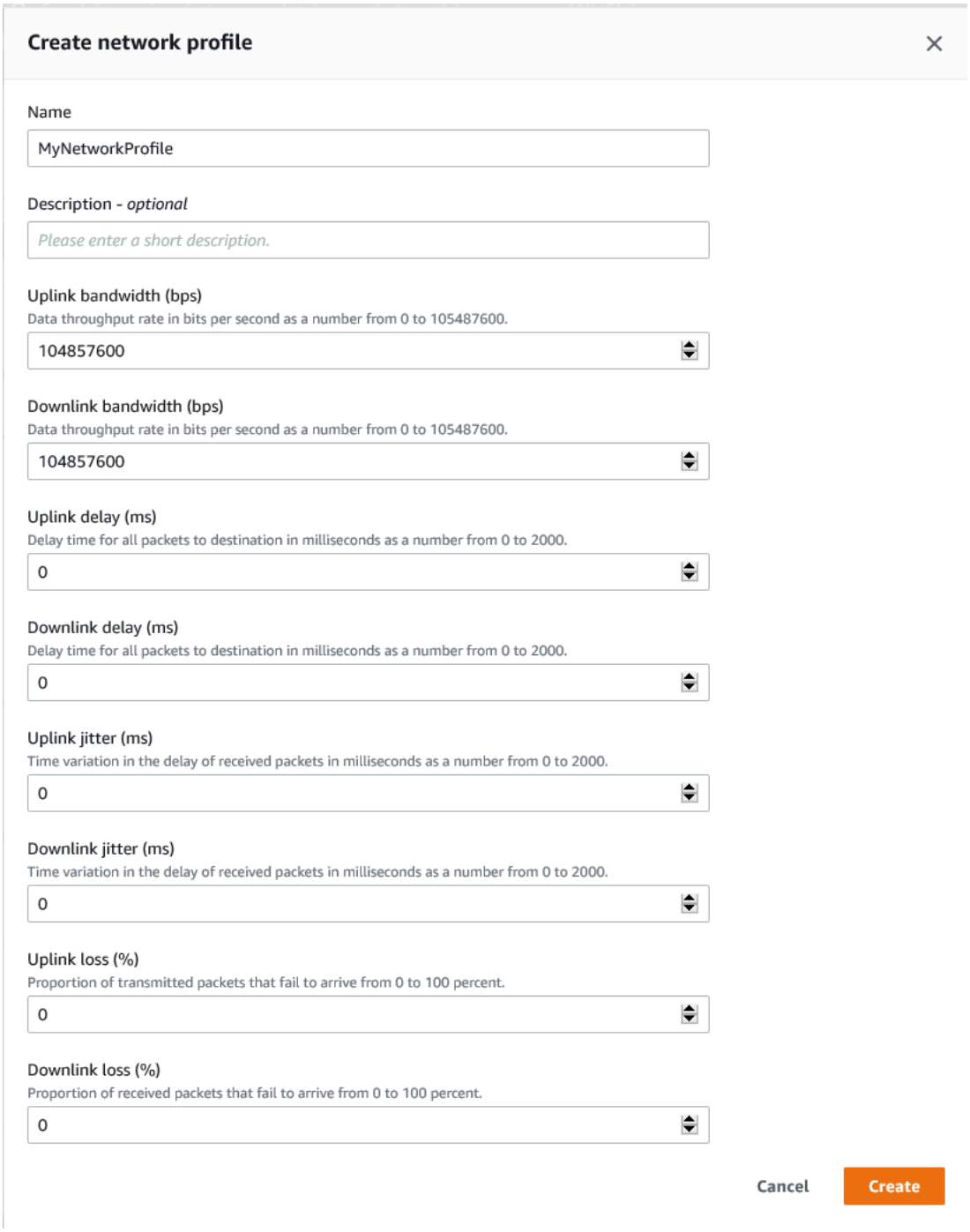


6. Wählen Sie Weiter aus.
7. Überprüfen und starten Sie Ihren Testlauf.

Erstellen Sie ein Netzwerkprofil

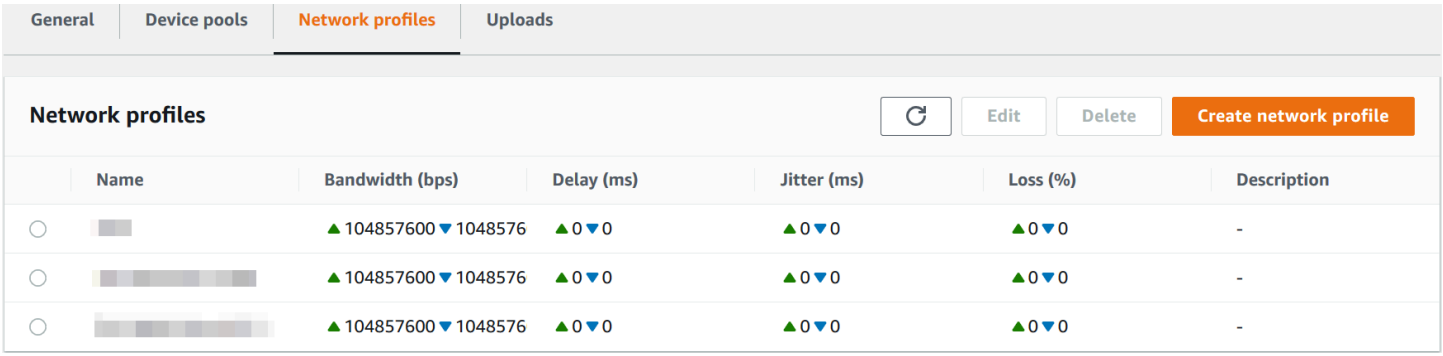
Wenn Sie einen Testlauf erstellen, können Sie ein neues Netzwerkprofil erstellen.

1. Wählen Sie Netzwerkprofil erstellen.



2. Geben Sie einen Namen und die Einstellungen für Ihr Netzwerkprofil ein.
3. Wählen Sie Erstellen aus.
4. Beenden Sie das Erstellen Ihres Testlaufs und starten Sie den Durchlauf.

Nachdem Sie ein Netzwerkprofil erstellt haben, können Sie es auf der Seite Project settings (Projekteinstellungen) anzeigen und verwalten.



Ändern Sie die Netzwerkbedingungen während des Tests

Sie können eine API von Ihrem Gerätehost aus aufrufen, indem Sie ein Framework wie Appium verwenden, um dynamische Netzwerkbedingungen wie reduzierte Bandbreite während Ihres Testlaufs zu simulieren. Weitere Informationen finden Sie unter CreateNetworkProfile.

Stoppen eines Laufs in AWS Device Farm

Sie möchten möglicherweise einen Testlauf stoppen, nachdem Sie ihn gestartet haben. Wenn Sie beispielsweise ein Problem während der Ausführung Ihrer Tests feststellen, möchten Sie den Testlauf möglicherweise mit einem aktualisierten Testskript erneut starten.

Sie können die Device Farm Farm-Konsole oder die API verwenden AWS CLI, um einen Lauf zu beenden.

Themen

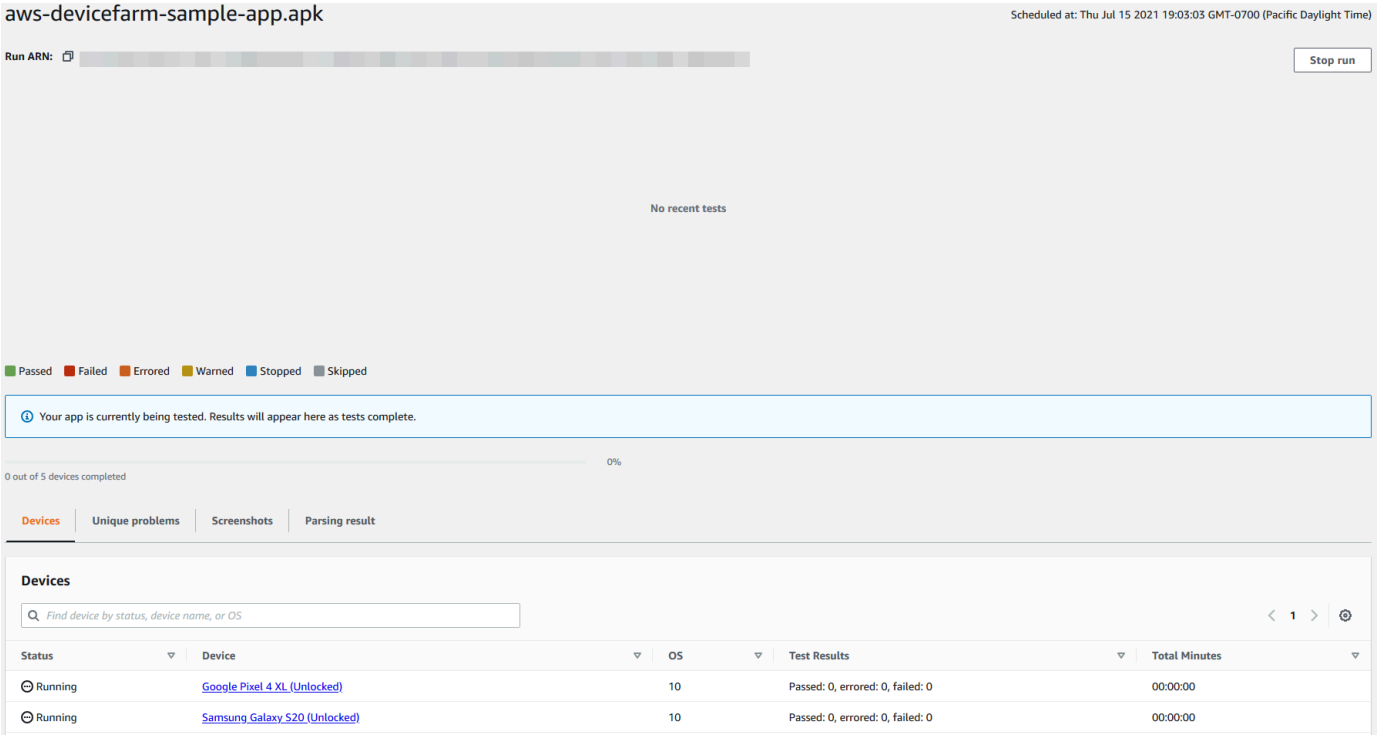
- Stoppen Sie einen Lauf (Konsole)
- Stoppen Sie einen Lauf ()AWS CLI
- Stoppen Sie einen Lauf (API)

Stoppen Sie einen Lauf (Konsole)

1. Melden Sie sich bei der Device Farm Farm-Konsole unter https://console.aws.amazon.com/devicefarm an.
2. Wählen Sie im Navigationsbereich Device Farm die Option Mobile Device Testing und dann Projects aus.
3. Wählen Sie das Projekt aus, für das Sie einen aktiven Testlauf durchgeführt haben.

4. Wählen Sie auf der Seite Automatisierte Tests den Testlauf aus.

   Das Symbol „Ausstehend“ oder „Wird ausgeführt“ sollte links neben dem Gerätenamen angezeigt werden.

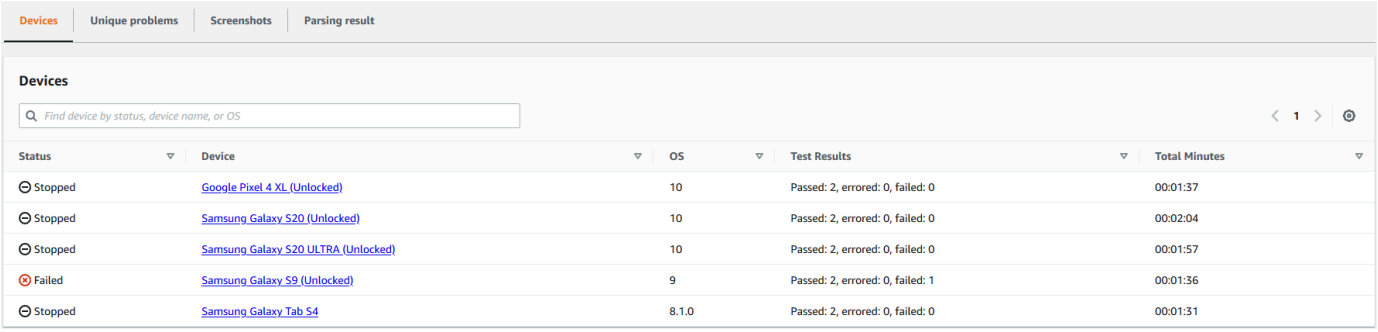


5. Wählen Sie Stop run (Testlauf stoppen).

   Nach kurzer Zeit erscheint neben dem Gerätenamen ein Symbol mit einem roten Kreis und einem Minus darin. Wenn der Lauf gestoppt wurde, wechselt die Farbe des Symbols von rot nach schwarz.

> **Important**
>
> Wenn ein Test bereits ausgeführt wurde, kann Device Farm ihn nicht beenden. Wenn ein Test läuft, stoppt Device Farm den Test. Die Gesamtzahl der Minuten, für die Sie Gebühren zahlen müssen, wird im Bereich Devices (Geräte) angezeigt. Darüber hinaus werden Ihnen auch die Gesamtminuten in Rechnung gestellt, die Device Farm benötigt, um die Setup Suite und die Teardown Suite auszuführen. Weitere Informationen finden Sie unter Device Farm – Preise.

   In der folgenden Abbildung wird ein Beispiel des Bereichs Devices (Geräte) angezeigt, nachdem der Testlauf erfolgreich gestoppt wurde.



Stoppen Sie einen Lauf ()AWS CLI

Sie können den folgenden Befehl ausführen, um den angegebenen Testlauf zu beenden. Dabei *myARN* handelt es sich um den Amazon-Ressourcennamen (ARN) des Testlaufs.

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

Die Ausgabe sollte folgendermaßen oder ähnlich aussehen:

```
{
    "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
        }
    }
}
```

Um den ARN Ihres Testlaufs zu erhalten, verwenden Sie den Befehl `list-runs`. Die Ausgabe sollte folgendermaßen oder ähnlich aussehen:

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

Informationen zur Verwendung von Device Farm mit dem AWS CLI finden Sie unterAWS CLI Referenz.

Stoppen Sie einen Lauf (API)

- Rufen Sie den StopRunVorgang zum Testlauf auf.

Hinweise zur Verwendung der Device Farm API finden Sie unterDevice Farm automatisieren.

Eine Liste von Läufen in AWS Device Farm anzeigen

Sie können die Device Farm Farm-Konsole oder die API verwenden AWS CLI, um eine Liste der Ausführungen für ein Projekt anzuzeigen.

Themen

Eine Liste der Läufe anzeigen (Konsole)

1. Melden Sie sich bei der Device Farm Farm-Konsole unter https://console.aws.amazon.com/devicefarm an.
2. Wählen Sie im Navigationsbereich Device Farm die Option Mobile Device Testing und dann Projects aus.
3. Wählen Sie in der Liste der Projekte das Projekt aus, das der Liste der Testläufe entspricht, die Sie anzeigen möchten.

> **Tip**
>
> Sie können die Suchleiste verwenden, um die Projektliste nach Namen zu filtern.

Eine Liste von Läufen anzeigen (AWS CLI)

- Führen Sie den Befehl list-runs aus.

  Um Informationen zu einer einzigen Ausführung anzuzeigen, führen Sie den Befehl get-run aus.

Informationen zur Verwendung von Device Farm mit dem AWS CLI finden Sie unterAWS CLI Referenz.

Eine Liste der Läufe anzeigen (API)

- Rufen Sie die `ListRuns`-API auf.

  Um Informationen zu einer einzigen Ausführung anzuzeigen, rufen Sie die `GetRun`-API auf.

Informationen zur Device Farm API finden Sie unterDevice Farm automatisieren.

Einen Gerätepool in AWS Device Farm erstellen

Sie können die Device Farm Farm-Konsole oder die API verwenden AWS CLI, um einen Gerätepool zu erstellen.

Themen

Voraussetzungen

- Erstellen Sie einen Lauf in der Device Farm Farm-Konsole. Folgen Sie den Anweisungen in Einen Testlauf in Device Farm erstellen. Wenn Sie die Seite Select devices (Geräte auswählen) erreichen, fahren Sie mit den Anweisungen in diesem Abschnitt fort.

Erstellen Sie einen Gerätepool (Konsole)

1. Wählen Sie auf der Seite Projekte Ihr Projekt aus. Wählen Sie auf der Seite mit den Projektdetails die Option Projekteinstellungen aus. Wählen Sie auf der Registerkarte Gerätepools die Option Gerätepool erstellen aus.
2. Geben Sie unter Name einen Namen ein, an dem Sie den Gerätepool leicht erkennen können.
3. Geben Sie unter Description (Beschreibung) eine Beschreibung ein, an der Sie den Gerätepool leicht erkennen können.

4. Wenn Sie eine oder mehrere Auswahlkriterien für die Geräte in diesem Gerätepool verwenden möchten, führen Sie die folgenden Schritte aus:

   a. Wählen Sie Dynamischen Gerätepool erstellen.

   b. Wählen Sie Regel hinzufügen aus.

   c. Wählen Sie für Feld (erste Dropdownliste) eine der folgenden Optionen aus:

      - Um Geräte nach ihrem Herstellernamen anzuzeigen, wählen Sie Gerätehersteller aus.
      - Um Geräte nach ihrem Formfaktor (Tablet oder Telefon) einzubeziehen, wählen Sie „Formfaktor“.
      - Um Geräte nach ihrem Verfügbarkeitsstatus basierend auf der Auslastung einzubeziehen, wählen Sie Verfügbarkeit.
      - Um nur öffentliche oder private Geräte einzubeziehen, wählen Sie Flottenart.
      - Um Geräte nach ihrem Betriebssystem einzubeziehen, wählen Sie Plattform.
      - Einige Geräte verfügen über ein zusätzliches Etikett oder eine Beschreibung des Geräts. Du kannst Geräte anhand ihres Labelinhalts suchen, indem du Instanzbezeichnungen auswählst.
      - Um Geräte nach ihrer Betriebssystemversion einzubeziehen, wählen Sie Betriebssystemversion.
      - Um Geräte nach ihrem Modell einzubeziehen, wählen Sie Modell.

   d. Wählen Sie unter Operator (zweite Dropdownliste) eine logische Operation (EQUALS, CONTAINS usw.) aus, um Geräte basierend auf der Abfrage einzubeziehen. Sie könnten beispielsweise Geräte *`Availability EQUALS AVAILABLE`* einbeziehen, die derzeit den `Available` Status haben.

   e. Geben Sie unter Wert (dritte Dropdownliste) den Wert ein, den Sie für die Werte Feld und Operator angeben möchten, oder wählen Sie ihn aus. Die Werte sind je nach Ihrer Feldauswahl begrenzt. Wenn Sie beispielsweise „Plattform“ für „Feld“ wählen, sind nur die Optionen ANDROID und IOS verfügbar. Wenn Sie „Formfaktor“ für „Feld“ auswählen, sind ebenfalls nur die Optionen PHONE und TABLET verfügbar.

   f. Um eine weitere Regel hinzuzufügen, wählen Sie Regel hinzufügen aus.

      Nachdem Sie die erste Regel erstellt haben, wird in der Geräteliste das Kontrollkästchen neben jedem Gerät, das mit der Regel übereinstimmt, aktiviert. Wenn Sie weitere Regeln erstellt oder vorhandene Regeln geändert haben, wird in der Geräteliste das Kontrollkästchen neben den Geräten, die mit diesen kombinierten Regeln übereinstimmen,

aktiviert. Geräte mit aktivierten Kontrollkästchen sind im Gerätepool eingeschlossen. Geräte mit deaktivieren Kontrollkästchen sind ausgeschlossen.

g. Geben Sie unter Max. Geräte die Anzahl der Geräte ein, die Sie in Ihrem Gerätepool verwenden möchten. Wenn Sie die maximale Anzahl von Geräten nicht eingeben, wählt Device Farm alle Geräte in der Flotte aus, die den von Ihnen erstellten Regeln entsprechen. Um zusätzliche Gebühren zu vermeiden, setzen Sie diese Zahl auf einen Betrag, der Ihren tatsächlichen Anforderungen an die parallel Ausführung und die Gerätevielfalt entspricht.

h. Um eine Regel zu löschen, wählen Sie Regel entfernen.

5. Wenn Sie einzelne Geräte manuell ein- oder ausschließen möchten, gehen Sie wie folgt vor:

   a. Wählen Sie Statischen Gerätepool erstellen aus.

   b. Aktivieren oder deaktivieren Sie das Kästchen neben jedem Gerät. Sie können die Kontrollkästchen nur aktivieren oder deaktivieren, wenn keine Regeln angegeben wurden.

6. Wenn Sie alle angezeigten Geräte ein- oder ausschließen möchten, aktivieren oder deaktivieren Sie das Kontrollkästchen in der Spaltenüberschriftenzeile der Liste. Wenn Sie nur private Geräteinstanzen anzeigen möchten, wählen Sie Nur private Geräteinstanzen anzeigen.

   > **Important**
   >
   > Auch wenn Sie die Kontrollkästchen in der Spaltenüberschriftenzeile verwenden können, um die Liste der angezeigten Geräte zu ändern, bedeutet das nicht, dass die verbleibenden angezeigten Geräte die einzigen sind, die aus- oder ausgeschlossen sind. Um zu bestätigen, welche Geräte ein- oder ausgeschlossen sind, löschen Sie den Inhalt aller Felder in der Spaltenüberschriftenzeile. Durchsuchen Sie anschließend die Felder.

7. Wählen Sie Erstellen aus.

Erstellen Sie einen Gerätepool (AWS CLI)

> **Tip**
>
> Wenn Sie die maximale Anzahl von Geräten nicht eingeben, wählt Device Farm alle Geräte in der Flotte aus, die den von Ihnen erstellten Regeln entsprechen. Um zusätzliche Gebühren zu vermeiden, setzen Sie diese Zahl auf einen Betrag, der Ihren tatsächlichen Anforderungen an die parallel Ausführung und die Gerätevielfalt entspricht.

- Führen Sie den Befehl create-device-pool aus.

Informationen zur Verwendung von Device Farm mit dem AWS CLI finden Sie unterAWS CLI Referenz.

Erstellen Sie einen Gerätepool (API)

> **Tip**
>
> Wenn Sie die maximale Anzahl von Geräten nicht eingeben, wählt Device Farm alle Geräte in der Flotte aus, die den von Ihnen erstellten Regeln entsprechen. Um zusätzliche Gebühren zu vermeiden, setzen Sie diese Zahl auf einen Betrag, der Ihren tatsächlichen Anforderungen an die parallel Ausführung und die Gerätevielfalt entspricht.

- Rufen Sie die `CreateDevicePool`-API auf.

Hinweise zur Verwendung der Device Farm API finden Sie unterDevice Farm automatisieren.

Analysieren von Testergebnissen in AWS Device Farm

In der Standardtestumgebung können Sie die Device Farm Farm-Konsole verwenden, um Berichte für jeden Test in Ihrem Testlauf anzuzeigen. Anhand der Berichte können Sie nachvollziehen, welche Tests bestanden oder nicht bestanden haben. Außerdem erhalten Sie Informationen zur Leistung und zum Verhalten Ihrer App in verschiedenen Gerätekonfigurationen.

Device Farm sammelt auch andere Artefakte wie Dateien, Protokolle und Bilder, die Sie herunterladen können, wenn Ihr Testlauf abgeschlossen ist. Mithilfe dieser Informationen können Sie analysieren, wie sich Ihre App auf echten Geräten verhält, Probleme oder Bugs identifizieren und Probleme diagnostizieren.

Themen

- Testberichte in Device Farm anzeigen
- Artefakte in Device Farm herunterladen

Testberichte in Device Farm anzeigen

Verwenden Sie die Device Farm Farm-Konsole, um Ihre Testberichte anzuzeigen. Weitere Informationen finden Sie unter Berichte in AWS Device Farm.

Themen

Voraussetzungen

Richten Sie einen Testlauf ein und stellen Sie sicher, dass er abgeschlossen ist.

1. Um einen Testlauf zu erstellen, befolgen Sie die Anweisungen unter Einen Testlauf in Device Farm erstellen und kehren Sie dann zu dieser Seite zurück.
2. Stellen Sie sicher, dass der Testlauf abgeschlossen ist. Während Ihres Testlaufs zeigt die Device Farm Farm-Konsole ein Symbol

   für laufende Läufe an. Jedes Gerät, das gerade ausgeführt wird, beginnt ebenfalls mit dem Symbol „Ausstehend“ und wechselt dann zu Beginn des Tests zum

   Symbol „Wird ausgeführt“. Nach Abschluss jedes Tests wird neben dem Gerätenamen ein Testergebnissymbol angezeigt. Wenn alle Tests abgeschlossen sind, ändert sich das Symbol für ausstehende Tests neben dem Testlauf in ein Testergebnissymbol. Weitere Informationen finden Sie unter Status der Device Farm Farm-Testergebnisse.

Berichte anzeigen

Sie können die Ergebnisse Ihres Tests in der Device Farm Farm-Konsole anzeigen.

Themen

- Berichte der Testsuite anzeigen
- Anzeigen von Testberichten
- Protokollinformationen für ein Problem, ein Gerät, eine Suite oder einen Test in einem Bericht anzeigen

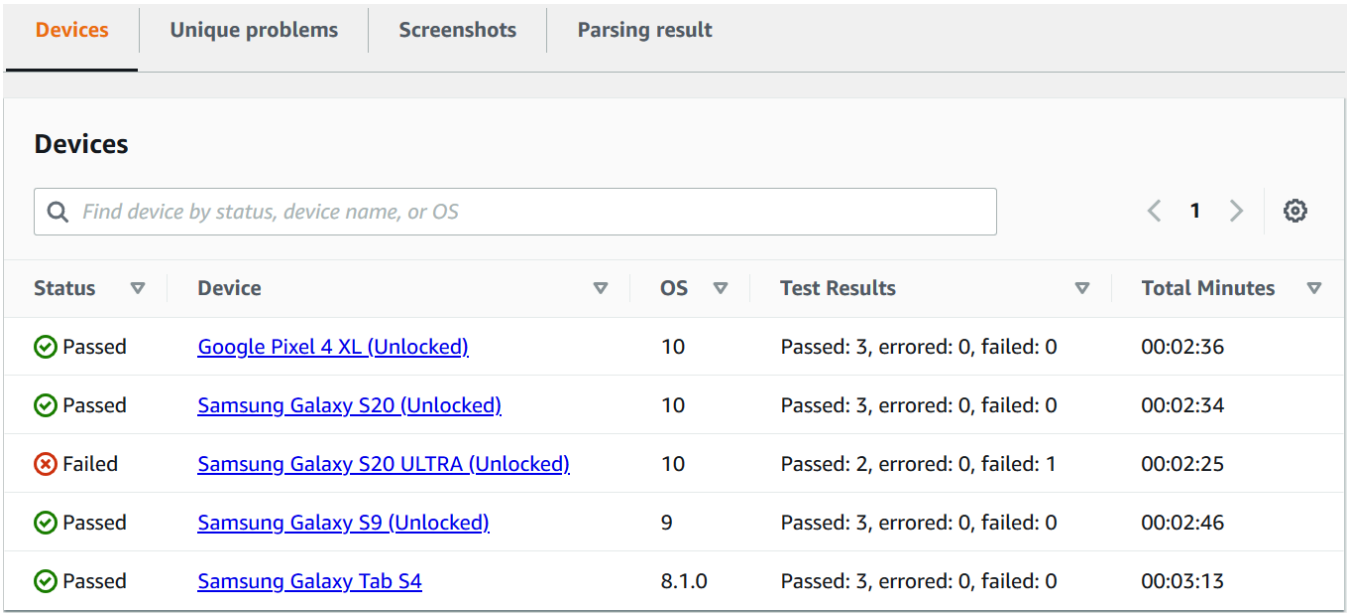
Sehen Sie sich die Seite mit der Zusammenfassung des Testlaufs an

1. Melden Sie sich bei der Device Farm Farm-Konsole unter https://console.aws.amazon.com/devicefarm an.
2. Wählen Sie im Navigationsbereich Mobile Device Testing und dann Projects aus.
3. Wählen Sie in der Liste der Projekte das Projekt für den Testlauf aus.

   > **Tip**
   >
   > Verwenden Sie die Suchleiste, um die Projektliste nach Namen zu filtern.

4. Wählen Sie einen abgeschlossenen Testlauf zum Anzeigen seiner Berichtsübersichtseite aus.
5. Die Testlauf-Übersichtsseite zeigt eine Übersicht Ihrer Testergebnisse an.

   - Im Bereich Unique problems (Eindeutige Probleme) werden eindeutige Warnungen und Fehler aufgeführt. Um eindeutige Probleme anzuzeigen, befolgen Sie die Anweisungen unter Einzigartige Problemberichte anzeigen.
   - Im Bereich Devices (Geräte) wird für jedes Gerät die Gesamtanzahl der Tests nach Ergebnissen angezeigt.



In diesem Beispiel gibt es mehrere Geräte. Im ersten Tabelleneintrag meldet das Google Pixel 4 XL-Gerät mit Android Version 10 drei erfolgreiche Tests, deren Ausführung 02:36 Minuten gedauert hat.

Um die Ergebnisse nach Gerät anzuzeigen, befolgen Sie die Anweisungen unter Geräteberichte anzeigen.

- Im Abschnitt Screenshots wird eine Liste aller Screenshots angezeigt, die Device Farm während des Laufs aufgenommen hat, gruppiert nach Geräten.
- Im Bereich Parsing-Ergebnisse können Sie das Parsing-Ergebnis herunterladen.

Einzigartige Problemberichte anzeigen

1. Wählen Sie unter Unique problems (Eindeutige Probleme) das Problem aus, das Sie anzeigen möchten.
2. Wählen Sie das Gerät aus. Der Bericht zeigt die Informationen zu dem Problem an.

   Im Bereich Video wird eine Videoaufnahme des Tests angezeigt, die Sie herunterladen können.

   Im Abschnitt Ergebnis wird das Ergebnis des Tests angezeigt. Der Status wird als Ergebnissymbol dargestellt. Weitere Informationen finden Sie unter Status eines einzelnen Tests.

   Im Abschnitt Protokolle werden alle Informationen angezeigt, die Device Farm während des Tests protokolliert hat. Sie können diese Informationen anzeigen, indem Sie die Anweisungen

unter Protokollinformationen für ein Problem, ein Gerät, eine Suite oder einen Test in einem Bericht anzeigen befolgen.

Auf der Registerkarte Dateien wird eine Liste aller mit dem Test verknüpften Dateien (z. B. Protokolldateien) angezeigt, die Sie herunterladen können. Sie können eine Datei herunterladen, indem Sie auf den Link in der Liste klicken.

Auf der Registerkarte Screenshots wird eine Liste aller Screenshots angezeigt, die Device Farm während des Tests aufgenommen hat.

Geräteberichte anzeigen

- Wählen Sie im Abschnitt Devices (Geräte) das Gerät aus.

  Im Bereich Video wird eine Videoaufnahme des Tests angezeigt, die Sie herunterladen können.

  Im Bereich Suiten wird eine Tabelle mit Informationen zu den Suiten für das Gerät angezeigt.

  In dieser Tabelle wird in der Spalte Testergebnisse die Anzahl der Tests nach Ergebnissen für jede der Testsuiten zusammengefasst, die auf dem Gerät ausgeführt wurden. Diese Daten haben auch eine grafische Komponente. Weitere Informationen finden Sie unter Status für mehrere Tests.

  Um die vollständigen Ergebnisse nach Suiten aufgeschlüsselt anzuzeigen, folgen Sie den Anweisungen unterBerichte der Testsuite anzeigen.

  Im Abschnitt Protokolle werden alle Informationen angezeigt, die Device Farm während der Ausführung für das Gerät protokolliert hat. Sie können diese Informationen anzeigen, indem Sie die Anweisungen unter Protokollinformationen für ein Problem, ein Gerät, eine Suite oder einen Test in einem Bericht anzeigen befolgen.

  Im Abschnitt Dateien werden eine Liste der Suites für das Gerät und alle zugehörigen Dateien (z. B. Protokolldateien) angezeigt, die Sie herunterladen können. Sie können eine Datei herunterladen, indem Sie auf den Link in der Liste klicken.

  Im Abschnitt Screenshots wird eine nach Suite gruppierte Liste aller Screenshots angezeigt, die Device Farm während der Ausführung für das Gerät aufgenommen hat.

Berichte der Testsuite anzeigen

1. Wählen Sie im Abschnitt Devices (Geräte) das Gerät aus.
2. Wählen Sie im Bereich Suiten die Suite aus der Tabelle aus.

   Im Bereich Video wird eine Videoaufnahme des Tests angezeigt, die Sie herunterladen können.

   Im Abschnitt Tests wird eine Tabelle mit Informationen zu den Tests in der Suite angezeigt.

   In der Tabelle wird in der Spalte Testergebnisse das Ergebnis angezeigt. Diese Daten haben auch eine grafische Komponente. Weitere Informationen finden Sie unter Status für mehrere Tests.

   Folgen Sie den Anweisungen unter, um die vollständigen Testergebnisse nach Tests einzusehenAnzeigen von Testberichten.

   Im Abschnitt Protokolle werden alle Informationen angezeigt, die Device Farm während der Ausführung der Suite protokolliert hat. Sie können diese Informationen anzeigen, indem Sie die Anweisungen unter Protokollinformationen für ein Problem, ein Gerät, eine Suite oder einen Test in einem Bericht anzeigen befolgen.

   Im Abschnitt Dateien werden eine Liste der Tests für die Suite und alle zugehörigen Dateien (z. B. Protokolldateien) angezeigt, die Sie herunterladen können. Sie können eine Datei herunterladen, indem Sie auf den Link in der Liste klicken.

   Im Abschnitt Screenshots wird eine Liste aller Screenshots angezeigt, die Device Farm während der Ausführung für die Suite aufgenommen hat, gruppiert nach Tests.

Anzeigen von Testberichten

1. Wählen Sie im Abschnitt Devices (Geräte) das Gerät aus.
2. Wählen Sie die Testreihe im Bereich Suites (Testreihen) aus.
3. Wählen Sie im Abschnitt Tests den Test aus.
4. Im Bereich Video wird eine Videoaufnahme des Tests angezeigt, die Sie herunterladen können.

   Im Abschnitt Ergebnis wird das Ergebnis des Tests angezeigt. Der Status wird als Ergebnissymbol dargestellt. Weitere Informationen finden Sie unter Status eines einzelnen Tests.

Im Abschnitt Protokolle werden alle Informationen angezeigt, die Device Farm während des Tests protokolliert hat. Sie können diese Informationen anzeigen, indem Sie die Anweisungen unter Protokollinformationen für ein Problem, ein Gerät, eine Suite oder einen Test in einem Bericht anzeigen befolgen.

Auf der Registerkarte Dateien wird eine Liste aller mit dem Test verknüpften Dateien (z. B. Protokolldateien) angezeigt, die Sie herunterladen können. Sie können eine Datei herunterladen, indem Sie auf den Link in der Liste klicken.

Auf der Registerkarte Screenshots wird eine Liste aller Screenshots angezeigt, die Device Farm während des Tests aufgenommen hat.

Protokollinformationen für ein Problem, ein Gerät, eine Suite oder einen Test in einem Bericht anzeigen

Im Abschnitt Protokolle werden die folgenden Informationen angezeigt:

- Unter Source (Quelle) wird die Quelle des jeweiligen Protokolleintrags angezeigt. Mögliche Werte sind:
  - Harness steht für einen Protokolleintrag, den Device Farm erstellt hat. Diese Protokolleinträge werden in der Regel während der Start- und Stopp-Ereignisse erstellt.
  - Device steht für einen Protokolleintrag, den das Gerät erstellt hat. Diese Protokolleinträge sind für Android mit logcat kompatibel. Für iOS sind diese Protokolleinträge mit syslog kompatibel.
  - Test stellt einen Protokolleintrag dar, der entweder von einem Test oder einem Test-Framework erstellt wurde.
- Time (Zeit) stellt die Zeit dar, die zwischen dem ersten Protokolleintrag und dem vorliegenden Protokolleintrag verstrichen ist. Die Zeit wird im *MM:SS.SSS* Format ausgedrückt, das *M* für Minuten und Sekunden *S* steht.
- PID stellt die Prozess-ID (PID) dar, mit der der Protokolleintrag erstellt wurde. Alle Protokolleinträge, die von einer bestimmten Anwendung und auf einem bestimmten Gerät erstellt wurden, weisen jeweils dieselbe PID auf.
- Level (Stufe) stellt die Protokollierungsstufe für den Protokolleintrag dar. `Logger.debug("This is a message!")` würde z. B. Einträge auf Level (Stufe) `Debug` erstellen. Mögliche Werte sind:
  - Warnung
  - Critical

  - Debuggen
  - Emergency (Notfall)
  - Fehler
  - Errored (Mit Fehlern)
  - Fehlgeschlagen
  - Info
  - Internal (Intern)
  - Notice (Hinweis)
  - Passed
  - Skipped
  - Angehalten
  - Verbose
  - Warned (Mit Warnungen)
  - Warnung
- Tag steht für beliebige Metadaten für den Protokolleintrag. Der Android-Systemlog (logcat) kann damit z. B. beschreiben, welcher Teil des Systems den Protokolleintrag erstellt hat (z. B. `ActivityManager`).
- Message (Meldung) stellt eine Meldung oder andere Information für den Protokolleintrag dar. `Logger.debug("Hello, World!")` würde z. B. Einträge mit einer Message (Meldung) von `"Hello, World!"` erstellen.

So zeigen Sie nur einen Teil der Informationen an:

- Um alle Protokolleinträge anzuzeigen, die einem Wert für eine bestimmte Spalte entsprechen, geben Sie den Wert in die Suchleiste ein. Um beispielsweise alle Protokolleinträge mit einem Quellwert von anzuzeigen`Harness`, geben Sie **`Harness`** in die Suchleiste ein.
- Sie können alle Zeichen in Feld einer Spaltenüberschrift entfernen, indem Sie im jeweiligen Feld auf das X klicken. Das Entfernen aller Zeichen aus einem Spaltenüberschriftenfeld entspricht der Eingabe ***** in dieses Spaltenüberschriftenfeld.

Um alle Protokollinformationen für das Gerät herunterzuladen, einschließlich aller von Ihnen ausgeführten Suiten und Tests, wählen Sie Protokolle herunterladen.

Status der Device Farm Farm-Testergebnisse

In der Device Farm Farm-Konsole werden Symbole angezeigt, mit denen Sie den Status Ihres abgeschlossenen Testlaufs schnell beurteilen können. Weitere Informationen zu Tests in Device Farm finden Sie unterBerichte in AWS Device Farm.

Themen

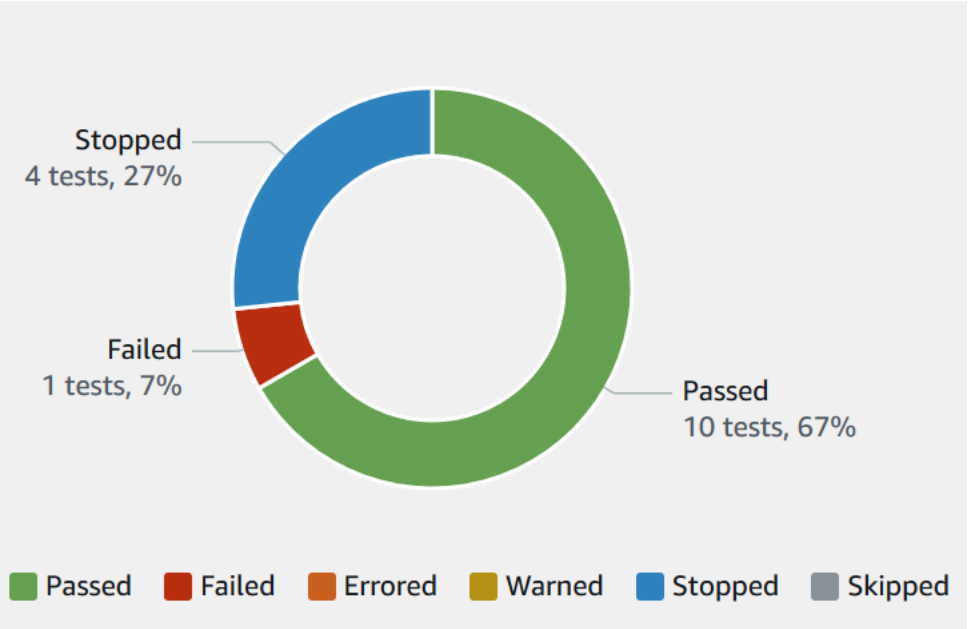
Status eines einzelnen Tests

Bei Berichten, die einen einzelnen Test beschreiben, zeigt Device Farm ein Symbol an, das den Status des Testergebnisses darstellt:

Description	Symbol
Der Test war erfolgreich.	
Der Test ist fehlgeschlagen.	
Device Farm hat den Test übersprungen.	
Der Test wurde beendet.	
Device Farm hat eine Warnung zurückgegeben.	
Device Farm hat einen Fehler zurückgegeben.	

Status für mehrere Tests

Wenn Sie einen abgeschlossenen Lauf wählen, zeigt Device Farm ein Übersichtsdiagramm an, das den Prozentsatz der Tests in verschiedenen Status zeigt.



Dieses Diagramm mit den Ergebnissen eines Testlaufs zeigt beispielsweise, dass der Testlauf 4 gestoppte Tests, 1 fehlgeschlagener Test und 10 erfolgreiche Tests umfasste.

Diagramme sind immer farblich gekennzeichnet und beschriftet.

Artefakte in Device Farm herunterladen

Device Farm sammelt Artefakte wie Berichte, Protokolldateien und Bilder für jeden Test in der Ausführung.

Sie können die während des Testlaufs erstellten Artefakte herunterladen:

Dateien

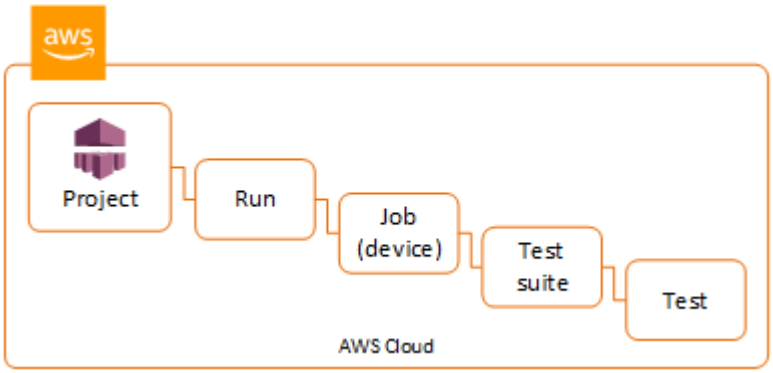
Während des Testlaufs generierte Dateien, einschließlich Device Farm Farm-Berichten. Weitere Informationen finden Sie unter Testberichte in Device Farm anzeigen.

Protokolle

Ausgabe aus jedem Test im Testlauf.

Screenshots

Bildschirmbilder, die für jeden Test im Testlauf aufgezeichnet wurden.



Artefakte herunterladen (Konsole)

1. Wählen Sie auf der Seite des Testlaufberichts unter Devices (Geräte) ein mobiles Gerät aus.
2. Um eine Datei herunterzuladen, wählen Sie sie unter Files (Dateien) aus.
3. Um die Protokolle aus dem Testlauf herunterzuladen, wählen Sie unter Logs (Protokolle) die Option Download logs (Protokolle herunterladen).
4. Um einen Screenshot herunterzuladen, wählen Sie unter Screenshots einen Screenshot aus.

Weitere Informationen zum Herunterladen von Artefakten in einer benutzerdefinierten Testumgebung finden Sie unter Artefakte werden in einer benutzerdefinierten Testumgebung heruntergeladen.

Artefakte herunterladen (AWS CLI)

Sie können das verwenden AWS CLI , um Ihre Testlauf-Artefakte aufzulisten.

Themen

- Schritt 1: Holen Sie sich Ihre Amazon Resource Names (ARN)
- Schritt 2: Listet eure Artefakte auf
- Schritt 3: Laden Sie Ihre Artefakte herunter

Schritt 1: Holen Sie sich Ihre Amazon Resource Names (ARN)

Sie können Ihre Artefakte nach Testlauf, Auftrag, Testreihe oder Test auflisten. Sie benötigen den zugehörigen ARN. Diese Tabelle zeigt den Eingabe-ARN für jeden der AWS CLI Listenbefehle:

AWS CLI Befehl auflisten	Erforderlicher ARN
list-projects	Dieser Befehl gibt alle Projekte zurück und erfordert keinen ARN.
list-runs	`project`
list-jobs	`run`
list-suites	`job`
list-tests	`suite`

Wenn Sie z. B. einen Test-ARN suchen, führen Sie list-tests unter Verwendung des Testreihen-ARN als Eingabeparameter aus.

Beispiel:

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

Die Antwort enthält den Test-ARN eines jeden Tests in der Testreihe.

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

Schritt 2: Listet eure Artefakte auf

Der Befehl AWS CLI list-artifacts gibt eine Liste von Artefakten wie Dateien, Screenshots und Logs zurück. Jedes Artefakt verfügt über eine URL, sodass Sie die Datei herunterladen können.

- Rufen Sie list-artifacts auf und geben Sie dabei eine Ausführung, einen Auftrag, eine Testreihe oder einen Test-ARN an. Geben Sie eine Art von FILE (Datei), LOG (Protokoll) oder SCREENSHOT an.

  Dieses Beispiel gibt eine Download-URL für jedes Artefakt zurück, das für einen bestimmten Test verfügbar ist:

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

  Die Antwort enthält eine Download-URL für jedes Artefakt.

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

Schritt 3: Laden Sie Ihre Artefakte herunter

- Laden Sie Ihr Artefakt mithilfe der URL aus dem vorherigen Schritt herunter. Dieses Beispiel verwendet curl zum Herunterladen einer Android Appium Java-Ausgabedatei:

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

Laden Sie Artefakte herunter (API)

Die Device Farm ListArtifactsAPI-Methode gibt eine Liste von Artefakten wie Dateien, Screenshots und Protokollen zurück. Jedes Artefakt verfügt über eine URL, sodass Sie die Datei herunterladen können.

Artefakte werden in einer benutzerdefinierten Testumgebung heruntergeladen

In einer benutzerdefinierten Testumgebung sammelt Device Farm Artefakte wie benutzerdefinierte Berichte, Protokolldateien und Bilder. Diese Artefakte sind für jedes Gerät im Testlauf verfügbar.

Sie können diese während des Testlaufs erstellten Artefakte herunterladen:

Ausgabe der Testspezifikation

Die Ausgabe von der Ausführung der Befehle in der YAML-Datei der Testspezifikation.

Kundenartefakte

Eine ZIP-Datei enthält die Artefakte des Testlaufs. Sie wird im Abschnitt artifacts: (Artefakte:) der YAML-Datei Ihrer Testspezifikation konfiguriert.

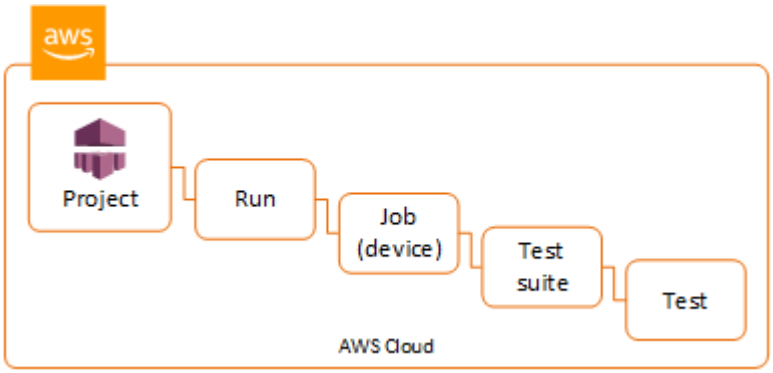
Shell-Skript der Test-Spezifikation

Eine intermediäre Shell-Skriptdatei, die aus der YAML-Datei erstellt wurde. Da die Shell-Skriptdatei im Testlauf verwendet wurde, kann sie für das Debugging der YAML-Datei genutzt werden.

Test-Spezifikationsdatei

Die im Testlauf verwendete YAML-Datei.

Weitere Informationen finden Sie unter Artefakte in Device Farm herunterladen.


aws
Project
Run
Job
(device)
Test
suite
Test
AWS Cloud

Taggen von AWS Device Farm Farm-Ressourcen

AWS Device Farm arbeitet mit der AWS Resource Groups Tagging API. Mit dieser API können Sie Ressourcen in Ihrem AWS -Konto mit Tags verwalten. Sie können Ressourcen, wie Projekte und Testläufe, Tags hinzufügen.

Sie können Tags verwenden, um:

- Organisieren Sie Ihre AWS-Kontorechnung, um Ihre eigene Kostenstruktur darzustellen. Dazu müssen Sie sich registrieren, um Ihre AWS-Kontorechnung mit Tag-Schlüsselwerten zu erhalten. Um dann die Kosten kombinierter Ressourcen anzuzeigen, organisieren Sie Ihre Fakturierungsinformationen nach Ressourcen mit gleichen Tag-Schlüsselwerten. Beispielsweise können Sie mehrere Ressourcen mit einem Anwendungsnamen markieren und dann Ihre Fakturierungsinformationen so organisieren, dass Sie die Gesamtkosten dieser Anwendung über mehrere Services hinweg sehen können. Weitere Informationen finden Sie unter Cost Allocation and Tagging in About AWS Billing and Cost Management.
- Steuern Sie den Zugriff über IAM-Richtlinien. Erstellen Sie dazu eine Richtlinie, die den Zugriff auf eine Ressource oder einen Satz von Ressourcen mithilfe einer Tag-Wertbedingung ermöglicht.
- Identifizieren und verwalten Sie Durchläufe, die bestimmte Eigenschaften haben, als Tags, z. B. den Zweig, der zum Testen verwendet wird.

Weitere Informationen zum Markieren von Ressourcen finden Sie im Whitepaper Bewährte Tagging-Methoden.

Topics

Taggen von -Ressourcen

Mit der AWS Resource Group Tagging API können Sie Ressourcen Tags hinzufügen, sie entfernen oder ändern. Weitere Informationen finden Sie in der API-Referenz der AWS-Ressourcengruppen-Markierung.

Verwenden Sie zum Taggen einer Ressource den Vorgang `TagResources` vom `resourcegroupstaggingapi`-Endpunkt aus. Für diesen Vorgang werden eine Liste der unterstützten Dienste und eine Liste ARNs von Schlüssel-Wert-Paaren verwendet. Der -Wert ist optional. Eine leere Zeichenfolge gibt an, dass für dieses Tag kein Wert vorhanden sein sollte. Das folgende Python-Beispiel kennzeichnet beispielsweise eine Reihe von Projekten ARNs mit dem Tag `build-config` mit dem Wert`release`:

```
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"})
```

Ein Tag-Wert ist nicht erforderlich. Um ein Tag ohne Wert festzulegen, verwenden Sie eine leere Zeichenfolge (""") für die Wertangabe. Ein Tag kann nur einen Wert haben. Jeder vorherige Wert, den ein Tag für eine Ressource hat, wird mit dem neuen Wert überschrieben.

Ressourcen anhand eines Tags nachschlagen

Verwenden Sie den `GetResources`-Vorgang vom `resourcegrouptaggingapi`-Endpunkt aus, um Ressourcen anhand ihrer Tags zu suchen. Dieser Vorgang verwendet eine Reihe von Filtern, von denen keiner erforderlich ist, und gibt die Ressourcen zurück, die den angegebenen Kriterien entsprechen. Ohne Filter werden alle getaggten Ressourcen zurückgegeben. Der `GetResources`-Vorgang ermöglicht das Filtern von Ressourcen basierend auf

- Tag-Wert
- Ressourcentyp (z. B. `devicefarm:run`)

Weitere Informationen finden Sie in der API-Referenz der AWS-Ressourcengruppen-Markierung.

Das folgende Beispiel sucht nach Device Farm Farm-Desktop-Browser-Testsitzungen (`devicefarm:testgrid-session`Ressourcen) mit dem Tag`stack`, die den Wert haben`production`:

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

Entfernen von Tags von Ressourcen

Verwenden Sie zum Entfernen eines Tags den `UntagResources`-Vorgang und geben Sie eine Liste der Ressourcen und die zu entfernenden Tags an:

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

Test-Frameworks und integrierte Tests in AWS Device Farm

In diesem Abschnitt wird die Device Farm Farm-Unterstützung für Test-Frameworks und integrierte Testtypen beschrieben.

Device Farm führt automatisierte Tests durch, indem Sie Ihre App und Tests in einen sicheren Amazon S3 S3-Bucket hochladen, der vom Service verwaltet wird. Nach dem Hochladen wird die zugrunde liegende Infrastruktur, einschließlich der vom Service verwalteten Testhosts, hochgefahren und die Tests werden parallel auf mehreren Geräten ausgeführt. Die Testergebnisse werden in einem vom Service verwalteten S3-Bucket gespeichert. Diese Architektur wird als serviceseitige Ausführung bezeichnet und ist eine schnelle und effiziente Möglichkeit, Tests auf Hosts auszuführen, die sich physisch in der Nähe des Geräts befinden, ohne die Testhostinfrastruktur selbst verwalten zu müssen. Dieser Ansatz eignet sich gut für Tests auf vielen Geräten unabhängig voneinander sowie für Tests im Kontext einer CI/CD Pipeline.

Weitere Informationen darüber, wie Device Farm Tests durchführt, finden Sie unterTestumgebungen in AWS Device Farm.

> **Note**
>
> Für Appium-Tester ziehen Sie es möglicherweise vor, Ihre Appium-Tests in Ihrer lokalen Umgebung auszuführen. Mit einer Fernzugriffssitzung können Sie clientseitige Appium-Tests ausführen. Weitere Informationen finden Sie unter clientseitige Appium-Tests.

Frameworks testen

Device Farm unterstützt die folgenden Frameworks für mobile Automatisierungstests:

Frameworks zum Testen von Android-Anwendungen

- Automatische Appium-Tests
- Instrumentierung

Frameworks zum Testen von iOS-Anwendungen

- Automatische Appium-Tests

- XCTest
- XCTest Benutzeroberfläche

Frameworks zum Testen von Webanwendungen

Webanwendungen werden durch Appium unterstützt. Weitere Informationen dazu, wie Sie Ihre Tests mit Appium verwenden, finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

Frameworks in einer benutzerdefinierten Testumgebung

Device Farm bietet keine Unterstützung für die Anpassung der Testumgebung für das XCTest Framework. Weitere Informationen finden Sie unter Benutzerdefinierte Testumgebungen in AWS Device Farm.

Unterstützung für Appium-Versionen

Für Tests, die in einer benutzerdefinierten Umgebung ausgeführt werden, unterstützt Device Farm Appium Version 1. Weitere Informationen finden Sie unter Testumgebungen in AWS Device Farm.

Integrierte Testtypen

Mit integrierten Tests können Sie Ihre Anwendung auf mehreren Geräten testen, ohne Testautomatisierungsskripts schreiben und verwalten zu müssen. Device Farm bietet einen integrierten Testtyp:

- Eingebaut: Fuzz (Android und iOS)

Automatisches Ausführen von Appium-Tests in Device Farm

> **Note**
>
> Diese Seite behandelt die Ausführung von Appium-Tests in der verwalteten serverseitigen Ausführungsumgebung von Device Farm. Informationen zum Ausführen von Appium-Tests in Ihrer lokalen clientseitigen Umgebung während einer Fernzugriffssitzung finden Sie unter Clientseitige Appium-Tests.

In diesem Abschnitt wird beschrieben, wie Sie Ihre Appium-Tests für die Ausführung in der verwalteten serverseitigen Umgebung von Device Farm konfigurieren, verpacken und hochladen. Appium ist ein Open-Source-Tool zur Automatisierung nativer und mobiler Webanwendungen. Weitere Informationen finden Sie unter Einführung in Appium auf der Appium-Website.

Eine Beispiel-App und Links zu funktionierenden Tests finden Sie unter Device Farm Farm-Beispiel-App für Android und Device Farm Farm-Beispiel-App für iOS auf GitHub.

Weitere Informationen zum Testen in Device Farm und zur serverseitigen Funktionsweise finden Sie unterTest-Frameworks und integrierte Tests in AWS Device Farm.

Eine Appium-Version auswählen

> **Note**
>
> Die Support bestimmter Appium-Versionen, Appium-Treiber oder Programmierung SDKs hängt vom Gerät und dem Testhost ab, die für den Testlauf ausgewählt wurden.

Device Farm Farm-Testhosts ist Appium vorinstalliert, um eine schnellere Einrichtung von Tests für einfachere Anwendungsfälle zu ermöglichen. Die Verwendung der Testspezifikationsdatei ermöglicht es Ihnen jedoch, bei Bedarf unterschiedliche Versionen von Appium zu installieren.

Szenario 1: Vorkonfigurierte Appium-Version

Device Farm ist mit verschiedenen Appium-Serverversionen vorkonfiguriert, die auf dem Testhost basieren. Der Host wird mit Tools geliefert, die die vorkonfigurierte Version mit dem Standardtreiber der Geräteplattform (UiAutomator2 für Android und XCUITest für iOS) aktivieren.

```
phases:
  install:
    commands:
      - export APPIUM_VERSION=2
      - devicefarm-cli use appium $APPIUM_VERSION
```

Eine Liste der unterstützten Software finden Sie im Thema unter. Unterstützte Software in benutzerdefinierten Testumgebungen

Szenario 2: Benutzerdefinierte Appium-Version

Um eine benutzerdefinierte Version von Appium auszuwählen, verwenden Sie den `npm` Befehl, um sie zu installieren. Das folgende Beispiel zeigt, wie Sie die neueste Version von Appium 2 installieren.

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

Szenario 3: Appium auf älteren iOS-Hosts

Auf dem Legacy-iOS-Testhost können Sie bestimmte Appium-Versionen mit auswählen. `avm` Um beispielsweise den `avm` Befehl zu verwenden, um die Appium-Serverversion auf einzustellen`2.1.2`, fügen Sie diese Befehle zu Ihrer YAML-Datei mit Testspezifikationen hinzu.

```
phases:
  install:
    commands:
      - export APPIUM_VERSION=2.1.2
      - avm $APPIUM_VERSION
```

Auswahl einer WebDriverAgent Version für iOS-Tests

Um Appium-Tests auf iOS-Geräten ausführen zu können, WebDriverAgent ist die Verwendung von erforderlich. Diese Anwendung muss signiert sein, um auf iOS-Geräten installiert werden zu können. Device Farm bietet vorsignierte Versionen davon WebDriverAgent , die während der Ausführung benutzerdefinierter Testumgebungen verfügbar sind.

Der folgende Codeausschnitt kann verwendet werden, um eine WebDriverAgent Version auf Device Farm in Ihrer Testspezifikationsdatei auszuwählen, die mit Ihrer XCTest UI-Treiberversion kompatibel ist.

```
phases:
  pre_test:
    commands:
      - |-
        APPIUM_DRIVER_VERSION=$(appium driver list --installed --json | jq -r
 ".xcuitest.version" | cut -d "." -f 1);
```

```
       CORRESPONDING_APPIUM_WDA=$(env | grep
"DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V${APPIUM_DRIVER_VERSION}")
       if [[ ! -z "$APPIUM_DRIVER_VERSION" ]] && [[ ! -z
"$CORRESPONDING_APPIUM_WDA" ]]; then
         echo "Using Device Farm's prebuilt WDA version ${APPIUM_DRIVER_VERSION}.x,
which corresponds with your driver";
         DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH=$(echo $CORRESPONDING_APPIUM_WDA |
cut -d "=" -f2)
       else
         LATEST_SUPPORTED_WDA_VERSION=$(env | grep
"DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V" | sort -V -r | head -n 1)
         echo "Unknown driver version $APPIUM_DRIVER_VERSION; falling back to the
Device Farm default version of $LATEST_SUPPORTED_WDA_VERSION";
         DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH=$(echo $LATEST_SUPPORTED_WDA_VERSION
| cut -d "=" -f2)
       fi;
```

Weitere Informationen zu finden Sie in der Dokumentation WebDriverAgent von Appium.

Integration von Appium-Tests mit Device Farm

Verwenden Sie die folgenden Anweisungen, um Appium-Tests in AWS Device Farm zu integrieren. Weitere Informationen zur Verwendung von Appium-Tests in Device Farm finden Sie unterAutomatisches Ausführen von Appium-Tests in Device Farm.

Konfigurieren Sie Ihr Appium-Testpaket

Anhand der folgenden Anweisungen können Sie Ihr Testpaket konfigurieren.

Java (JUnit)

1. Ändern Siepom.xml, um die Paketierung auf eine JAR-Datei festzulegen:

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

2. Ändern Sie pom.xml so, dass als Zielplattform für den Build Ihrer Tests in einer JAR-Datei maven-jar-plugin verwendet wird.

Das folgende Plugin baut Ihren Testquellcode (alles im `src/test` Verzeichnis) in eine JAR-Datei ein:

```
<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. Ändern Sie `pom.xml` so, dass die Option `maven-dependency-plugin` verwendet wird, um die Abhängigkeiten als JAR-Dateien zu erzeugen.

   Das folgende Plugin kopiert deine Abhängigkeiten in das `dependency-jars` Verzeichnis:

```
<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. Speichern Sie die folgende XML-Assembly-Struktur nach `src/main/assembly/zip.xml`.

Das folgende XML ist eine Assemblydefinition, die, wenn sie konfiguriert ist, Maven anweist, eine .zip-Datei zu erstellen, die alles enthält, was sich im Stammverzeichnis Ihres Build-Ausgabeverzeichnisses und im Verzeichnis befindet: `dependency-jars`

```
<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. Ändern Sie `pom.xml` so, dass `maven-assembly-plugin` die Tests und alle Abhängigkeiten zusammen in einer ZIP-Datei paketiert.

   Das folgende Plug-in verwendet die Assembly-Struktur oben, um jedes Mal, wenn der Befehl mvn package ausgeführt wird, eine ZIP-Datei mit dem Namen `zip-with-dependencies` im Build-Ausgabeverzeichnis zu erstellen:

```
<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**
>
> Wenn Sie eine Fehlermeldung erhalten, dass in Version 1.3 Anmerkungen nicht unterstützt werden, fügen Sie Folgendes in `pom.xml` ein:
>
> ```
> <plugin>
>   <artifactId>maven-compiler-plugin</artifactId>
>   <configuration>
>     <source>1.7</source>
>     <target>1.7</target>
>   </configuration>
> </plugin>
> ```

Java (TestNG)

1. Ändern Sie`pom.xml`, um die Paketierung auf eine JAR-Datei festzulegen:

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

2. Ändern Sie `pom.xml` so, dass als Zielplattform für den Build Ihrer Tests in einer JAR-Datei `maven-jar-plugin` verwendet wird.

   Das folgende Plugin baut Ihren Testquellcode (alles im `src/test` Verzeichnis) in eine JAR-Datei ein:

```
<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. Ändern Sie `pom.xml` so, dass die Option `maven-dependency-plugin` verwendet wird, um die Abhängigkeiten als JAR-Dateien zu erzeugen.

   Das folgende Plugin kopiert deine Abhängigkeiten in das `dependency-jars` Verzeichnis:

```
<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. Speichern Sie die folgende XML-Assembly-Struktur nach `src/main/assembly/zip.xml`.

   Das folgende XML ist eine Assemblydefinition, die, wenn sie konfiguriert ist, Maven anweist, eine .zip-Datei zu erstellen, die alles enthält, was sich im Stammverzeichnis Ihres Build-Ausgabeverzeichnisses und im Verzeichnis befindet: `dependency-jars`

```
<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. Ändern Sie `pom.xml` so, dass `maven-assembly-plugin` die Tests und alle Abhängigkeiten zusammen in einer ZIP-Datei paketiert.

   Das folgende Plug-in verwendet die Assembly-Struktur oben, um jedes Mal, wenn der Befehl mvn package ausgeführt wird, eine ZIP-Datei mit dem Namen `zip-with-dependencies` im Build-Ausgabeverzeichnis zu erstellen:

```
<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**
>
> Wenn Sie eine Fehlermeldung erhalten, dass in Version 1.3 Anmerkungen nicht unterstützt werden, fügen Sie Folgendes in `pom.xml` ein:
>
> ```
> <plugin>
>   <artifactId>maven-compiler-plugin</artifactId>
>   <configuration>
>     <source>1.7</source>
>     <target>1.7</target>
>   </configuration>
> </plugin>
> ```

Node.JS

Um Ihre Appium Node.js Tests zu packen und auf Device Farm hochzuladen, müssen Sie Folgendes auf Ihrem lokalen Computer installieren:

- Node Version Manager (nvm)

Verwenden Sie dieses Tool, wenn Sie Ihre Tests entwickeln und Testpakete erstellen, damit keine unnötigen Abhängigkeiten in Ihr Testpaket eingeschlossen werden.

- Node.js
- npm-bundle (global installiert)

1. Stellen Sie sicher, dass nvm vorhanden ist.

   ```
   command -v nvm
   ```

   Sie sollten als Ausgabe `nvm` sehen.

   Weitere Informationen finden Sie unter nvm on. GitHub
2. Führen Sie diesen Befehl aus, um Node.js zu installieren:

   ```
   nvm install node
   ```

   Sie können eine bestimmte Version von Node.js angeben:

   ```
   nvm install 11.4.0
   ```
3. Stellen Sie sicher, dass die richtige Version von Node verwendet wird:

   ```
   node -v
   ```
4. Installieren Sie npm-bundle global:

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

Python

1. Es wird ausdrücklich empfohlen, das Python-Tool virtualenv für die Entwicklung und Erstellen von Testpaketen einzurichten und zu verwenden, um zu vermeiden, dass unnötige Abhängigkeiten in Ihr App-Paket aufgenommen werden.

   ```
   $ virtualenv workspace
   $ cd workspace
   ```

```
$ source bin/activate
```

> **Tip**
>
> - Erstellen Sie keine virtuelle Python-Umgebung mit der Option `--system-site-packages`, da in diesem Fall Pakete aus Ihrem globalen Verzeichnis „site-packages“ vererbt werden. Dies kann dazu führen, dass Abhängigkeiten in Ihre virtuelle Umgebung eingeschlossen werden, die von Ihren Tests nicht benötigt werden.
> - Sie sollten auch sicherstellen, dass Ihre Tests keine Abhängigkeiten von nativen Bibliotheken verwenden, da nicht sicher ist, ob diese nativen Bibliotheken in der Instance vorhanden sind, in der die Tests ausgeführt werden.

2. Installieren Sie py.test in Ihrer virtuellen Umgebung.

```
$ pip install pytest
```

3. Installieren Sie den Appium Python-Client in Ihrer virtuellen Umgebung.

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

4. Sofern Sie im benutzerdefinierten Modus keinen anderen Pfad angeben, erwartet Device Farm, dass Ihre Tests in gespeichert werden`tests/`. Sie können `find` verwenden, um alle Dateien in einem Ordner anzuzeigen:

```
$ find tests/
```

   Stellen Sie sicher, dass diese Dateien Testsuiten enthalten, die Sie auf Device Farm ausführen möchten

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

5. Führen Sie diesen Befehl vom Workspace-Ordner in Ihrer virtuellen Umgebung aus, um eine Liste Ihrer Tests anzuzeigen, ohne sie auszuführen.

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

Vergewissern Sie sich, dass in der Ausgabe die Tests angezeigt werden, die Sie auf Device Farm ausführen möchten.

6. Bereinigen Sie alle zwischengespeicherten Dateien unter Ihrem „tests/“-Ordner:

```
$ 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. Führen Sie in Ihrem Workspace den folgenden Befehl zum Generieren der requirements.txt-Datei aus:

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

Ruby

Um Ihre Appium Ruby-Tests zu packen und auf Device Farm hochzuladen, müssen Sie Folgendes auf Ihrem lokalen Computer installieren:

- Ruby Version Manager (RVM)

  Verwenden Sie dieses Befehlszeilen-Tool, wenn Sie Ihre Tests entwickeln und Testpakete erstellen, um zu vermeiden, dass unnötige Abhängigkeiten in Ihr Testpaket eingeschlossen werden.
- Ruby
- Bundler (Dieses Gem wird in der Regel mit Ruby installiert.)

1. Installieren Sie die erforderlichen Schlüssel, RVM und Ruby. Anweisungen finden Sie unter Installing RVM auf der RVM-Website.

   Nachdem die Installation abgeschlossen wurde, laden Sie Ihr Terminal erneut, indem Sie sich abmelden und dann erneut wieder anmelden.

   > **Note**
   >
   > RVM wird als Funktion nur für die bash-Shell geladen.

2. Stellen Sie sicher, dass rvm korrekt installiert ist.

```
command -v rvm
```

   Sie sollten als Ausgabe `rvm` sehen.

3. Wenn Sie beispielsweise eine bestimmte Version von Ruby installieren möchten*2.5.3*, führen Sie den folgenden Befehl aus:

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

   Stellen Sie sicher, dass Sie die angeforderte Version von Ruby verwenden:

```
ruby -v
```

4. Konfigurieren Sie den Bundler so, dass er Pakete für Ihre gewünschten Testplattformen kompiliert:

```
bundle config specific_platform true
```

5. Aktualisieren Sie Ihre .lock-Datei, um die Plattformen hinzuzufügen, die für die Ausführung von Tests benötigt werden.

   - Wenn Sie Tests für die Ausführung auf Android-Geräten kompilieren, führen Sie diesen Befehl aus, um das Gemfile so zu konfigurieren, dass es Abhängigkeiten für den Android-Testhost verwendet:

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

   - Wenn Sie Tests für die Ausführung auf iOS-Geräten kompilieren, führen Sie diesen Befehl aus, um das Gemfile so zu konfigurieren, dass es Abhängigkeiten für den iOS-Testhost verwendet:

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

6. Das Gem bundler ist normalerweise standardmäßig installiert. Wenn dies nicht der Fall ist, installieren Sie es:

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

Erstellen Sie eine komprimierte Testpaketdatei

> **Warning**
>
> In Device Farm ist die Ordnerstruktur der Dateien in Ihrem komprimierten Testpaket wichtig, und einige Archivierungstools ändern die Struktur Ihrer ZIP-Datei implizit. Wir empfehlen, dass Sie die unten angegebenen Befehlszeilenprogramme verwenden, anstatt die in den Dateimanager Ihres lokalen Desktops integrierten Archivierungsprogramme (wie Finder oder Windows Explorer) zu verwenden.

Bündeln Sie nun Ihre Tests für die Device Farm.

Java (JUnit)

Erstellen und verpacken Sie Ihre Tests:

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

Als Ergebnis wird die Datei `zip-with-dependencies.zip` erstellt. Dies ist Ihr Testpaket.

Java (TestNG)

Erstellen und verpacken Sie Ihre Tests:

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

Als Ergebnis wird die Datei `zip-with-dependencies.zip` erstellt. Dies ist Ihr Testpaket.

Node.JS

1. Überprüfen Sie Ihr Projekt.

   Stellen Sie sicher, dass Sie sich im Stammverzeichnis Ihres Projekts befinden. Sie sehen `package.json` im Stammverzeichnis.
2. Führen Sie diesen Befehl aus, um Ihre lokalen Abhängigkeiten zu installieren.

   ```
   npm install
   ```

   Dieser Befehl erstellt außerdem den Ordner `node_modules` in Ihrem aktuellen Verzeichnis.

> **Note**
>
> Zu diesem Zeitpunkt sollten Sie in der Lage sein, Ihre Tests lokal auszuführen.

3. Führen Sie diesen Befehl aus, um aus den Dateien in Ihrem aktuellen Ordner ein Testpaket in Form einer *.tgz-Datei zu erstellen. Die Datei wird unter Verwendung der Eigenschaft `name` in Ihrer `package.json`-Datei benannt.

```
npm-bundle
```

   Diese Tarball-Datei (.tgz) enthält Ihren gesamten Code und alle Abhängigkeiten.

4. Führen Sie diesen Befehl aus, um den im vorherigen Schritt erstellten Tarball (*.tgz-Datei) in einem einzigen ZIP-Archiv zu bündeln:

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

   Dies ist die `MyTests.zip` Datei, die Sie im folgenden Verfahren auf Device Farm hochladen.

Python

Python 2

Generieren Sie mit pip ein Archiv der erforderlichen Python-Pakete (als „Wheelhouse“ bezeichnet):

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

Packen Sie Ihr Wheelhouse, Ihre Tests und Ihre Pip-Anforderungen in ein Zip-Archiv für Device Farm:

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

Python 3

Packen Sie Ihre Tests und Pip-Anforderungen in eine ZIP-Datei:

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

Ruby

1. Führen Sie diesen Befehl aus, um eine virtuelle Ruby-Umgebung zu erstellen:

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

2. Führen Sie diesen Befehl aus, um die Umgebung, die Sie gerade erstellt haben, zu verwenden:

```
rvm gemset use myGemset
```

3. Überprüfen Sie den Quellcode.

   Stellen Sie sicher, dass Sie sich im Stammverzeichnis Ihres Projekts befinden. Sie sehen `Gemfile` im Stammverzeichnis.

4. Führen Sie diesen Befehl aus, um Ihre lokalen Abhängigkeiten und alle Gems aus der `Gemfile` zu installieren:

```
bundle install
```

> **Note**
>
> Zu diesem Zeitpunkt sollten Sie in der Lage sein, Ihre Tests lokal auszuführen. Mit diesem Befehl können Sie einen lokalen Test ausführen:
>
> ```
> bundle exec $test_command
> ```

5. Verpacken Sie Ihre Gems im Ordner `vendor/cache`.

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

6. Führen Sie den folgenden Befehl aus, um Ihren Quellcode zusammen mit allen Ihren Abhängigkeiten in einem einzigen ZIP-Archiv zu bündeln:

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

Dies ist die `MyTests.zip` Datei, die Sie im folgenden Verfahren auf Device Farm hochladen.

Laden Sie Ihr Testpaket auf Device Farm hoch

Sie können die Device Farm Farm-Konsole verwenden, um Ihre Tests hochzuladen.

1. Melden Sie sich bei der Device Farm Farm-Konsole unter https://console.aws.amazon.com/devicefarm an.
2. Wählen Sie im Navigationsbereich Device Farm die Option Mobile Device Testing und dann Projects aus.
3. Wenn Sie ein neuer Benutzer sind, wählen Sie „Neues Projekt“, geben Sie einen Namen für das Projekt ein und wählen Sie dann „Senden“.

   Wenn Sie bereits ein Projekt haben, können Sie es auswählen, um Ihre Tests darauf hochzuladen.
4. Öffnen Sie Ihr Projekt und wählen Sie dann Create run aus.
5. Geben Sie Ihrem Test unter Ausführungseinstellungen einen passenden Namen. Dieser kann eine beliebige Kombination aus Leerzeichen oder Satzzeichen enthalten.
6. Für native Android- und iOS-Tests

   Wählen Sie unter Einstellungen ausführen die Option Android-App aus, wenn Sie eine Android-Anwendung (.apk) testen, oder wählen Sie iOS-App, wenn Sie eine iOS-Anwendung (.ipa) testen. Wählen Sie dann unter App auswählen die Option Eigene App hochladen aus, um das verteilbare Paket Ihrer Anwendung hochzuladen.

   > **Note**
   >
   > Die Datei muss entweder eine Android `.apk`- oder eine iOS `.ipa`-Datei sein. iOS-Anwendungen müssen für echte Geräte erstellt werden, nicht für den Simulator.

   Für Tests von mobilen Webanwendungen

   Wählen Sie unter Einstellungen ausführen die Option Web-App aus.
7. Wählen Sie unter Test konfigurieren im Abschnitt Testframework auswählen das Appium-Framework aus, mit dem Sie testen, und laden Sie dann Ihr eigenes Testpaket hoch.

8. Navigieren Sie zu der ZIP-Datei, die Ihre Tests enthält, und wählen Sie diese aus. Die ZIP-Datei muss dem Format entsprechen, das unter Konfigurieren Sie Ihr Appium-Testpaket beschrieben wird.
9. Folgen Sie den Anweisungen, um Geräte auszuwählen und den Testlauf zu starten. Weitere Informationen finden Sie unter Einen Testlauf in Device Farm erstellen.

> **Note**
>
> Device Farm ändert Appium-Tests nicht.

Machen Sie Screenshots Ihrer Tests (optional)

Sie können im Rahmen Ihrer Tests Screenshots erstellen.

Device Farm setzt das `DEVICEFARM_SCREENSHOT_PATH`-Attribut auf einen vollqualifizierten Pfad auf dem lokalen Dateisystem. Device Farm erwartet, dass Appium-Screenshots unter diesem Pfad gespeichert werden. Das testspezifische Verzeichnis, in dem die Screenshots gespeichert werden, wird zur Laufzeit definiert. Die Screenshots werden automatisch in Ihre Device Farm-Berichte eingebunden. Sie können die Screenshots in der Device Farm-Konsole im Bereich Screenshots anzeigen.

Weitere Informationen zum Erstellen von Screenshots in Appium-Tests finden Sie unter Take Screenshot (Screenshot erstellen) in der Appium-API-Dokumentation.

Android-Tests in der AWS Device Farm

Device Farm bietet Unterstützung für verschiedene Automatisierungstesttypen für Android-Geräte sowie zwei integrierte Tests.

Weitere Informationen zum Testen in Device Farm finden Sie unterTest-Frameworks und integrierte Tests in AWS Device Farm.

Frameworks zum Testen von Android-Anwendungen

Die folgenden benutzerdefinierten Tests stehen für Android-Geräte zur Verfügung.

- Automatische Appium-Tests

- Instrumentierung

Integrierte Testtypen für Android

Für Android-Geräte ist ein integrierter Testtyp verfügbar:

- Eingebaut: Fuzz (Android und iOS)

Instrumentierung für Android und AWS Device Farm

Device Farm bietet Unterstützung für Instrumentation (JUnit, Espresso, Robotium oder andere instrumentationsbasierte Tests) für Android.

Device Farm bietet auch eine Android-Beispielanwendung und Links zu Arbeitstests in drei Android-Automatisierungsframeworks, darunter Instrumentation (Espresso). Die Device Farm Farm-Beispiel-App für Android steht unter zum Download bereit GitHub.

Weitere Informationen zum Testen in Device Farm finden Sie unterTest-Frameworks und integrierte Tests in AWS Device Farm.

Themen

- Was ist Instrumentierung?
- Überlegungen zu Android-Instrumentierungstests
- Testanalyse im Standardmodus
- Integrieren von Android Instrumentation in Device Farm

Was ist Instrumentierung?

Mit der Android-Instrumentierung können Sie Rückrufmethoden in Ihrem Testcode aufrufen. So können Sie den Lebenszyklus einer Komponente schrittweise durchlaufen, so als ob Sie die Komponente debuggen. Weitere Informationen finden Sie unter Instrumentierte Tests im Abschnitt Testtypen und -orte der Dokumentation zu den Android Developer Tools.

Überlegungen zu Android-Instrumentierungstests

Beachten Sie bei der Verwendung der Android-Instrumentierung die folgenden Empfehlungen und Hinweise.

Überprüfen Sie die Kompatibilität mit dem Android-Betriebssystem

Überprüfen Sie in der Android-Dokumentation, ob Instrumentation mit Ihrer Android-Betriebssystemversion kompatibel ist.

Wird von der Befehlszeile aus ausgeführt

Um Instrumentierungstests über die Befehlszeile auszuführen, folgen Sie bitte der Android-Dokumentation.

System Animations (Systemanimationen)

Gemäß der Android-Dokumentation für Espresso-Tests wird empfohlen, die Systemanimationen beim Testen auf echten Geräten auszuschalten. Device Farm deaktiviert automatisch die Einstellungen Window Animation Scale, Transition Animation Scale und Animator Duration Scale, wenn es mit dem Test-Runner android.support.test.runner.Android JUnit Runner Instrumentation Test Runner ausgeführt wird.

Test Recorders (Test-Aufzeichnungen)

Device Farm unterstützt Frameworks wie Robotium, die über record-and-playback Skripttools verfügen.

Testanalyse im Standardmodus

Im Standardmodus einer Ausführung analysiert Device Farm Ihre Testsuite und identifiziert die eindeutigen Testklassen und Methoden, die ausgeführt werden sollen. Dies erfolgt über ein Tool namens Dex Test Parser.

Wenn eine APK-Datei mit Android-Instrumentierung als Eingabe angegeben wird, gibt der Parser die vollständig qualifizierten Methodennamen der Tests zurück, die den Konventionen JUnit 3 und JUnit 4 entsprechen.

Um dies in einer lokalen Umgebung zu testen:

1. Laden Sie die `dex-test-parser`Binärdatei herunter.
2. Führen Sie den folgenden Befehl aus, um die Liste der Testmethoden abzurufen, die auf Device Farm ausgeführt werden:

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

Integrieren von Android Instrumentation in Device Farm

> **Note**
>
> Verwenden Sie die folgenden Anweisungen, um Android-Instrumentierungstests in AWS Device Farm zu integrieren. Weitere Informationen zur Verwendung von Instrumentierungstests in Device Farm finden Sie unterInstrumentierung für Android und AWS Device Farm.

Laden Sie Ihre Android-Instrumentierungstests hoch

Verwenden Sie die Device Farm Farm-Konsole, um Ihre Tests hochzuladen.

1. Melden Sie sich bei der Device Farm Farm-Konsole unter https://console.aws.amazon.com/devicefarm an.
2. Wählen Sie im Navigationsbereich Device Farm die Option Mobile Device Testing und dann Projects aus.
3. Wählen Sie in der Projektliste das Projekt aus, in das Sie Ihre Tests hochladen möchten.

   > **Tip**
   >
   > Sie können die Suchleiste verwenden, um die Projektliste nach Namen zu filtern. Befolgen Sie die Anweisungen unter Ein Projekt in AWS Device Farm erstellen, um ein neues Projekt zu erstellen.

4. Wählen Sie Lauf erstellen aus.
5. Wählen Sie unter App auswählen im Abschnitt App-Auswahloptionen die Option Eigene App hochladen aus.
6. Navigieren Sie zu der Datei mit Ihrer Android-Anwendung, und wählen Sie diese aus. Es muss sich dabei um eine APK-Datei handeln.
7. Wählen Sie unter Test konfigurieren im Abschnitt Testframework auswählen die Option Instrumentation und dann Datei auswählen aus.
8. Navigieren Sie zu der APK-Datei, die Ihre Tests enthält, und wählen Sie diese aus.
9. Folgen Sie den verbleibenden Anweisungen, um Geräte auszuwählen und den Testlauf zu starten.

(Optional) Machen Sie Screenshots von Android-Instrumentierungstests

Sie können im Rahmen Ihrer Android-Instrumentierungstests Screenshots erstellen.

Rufen Sie eine der folgenden Methoden auf, um Screenshots zu erstellen:

- Rufen Sie für Robotium die Methode `takeScreenShot` auf (z. B. `solo.takeScreenShot();`).
- Rufen Sie für Spoon die Methode `screenshot` auf, z. B.:

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

Während eines Testlaufs ruft Device Farm Screenshots von den folgenden Speicherorten auf den Geräten ab, sofern sie vorhanden sind, und fügt sie dann den Testberichten hinzu:

- `/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`*

iOS-Tests in der AWS Device Farm

Device Farm bietet Unterstützung für verschiedene Automatisierungstesttypen für iOS-Geräte sowie einen integrierten Test.

Weitere Informationen zum Testen in Device Farm finden Sie unterTest-Frameworks und integrierte Tests in AWS Device Farm.

Frameworks zum Testen von iOS-Anwendungen

Die folgenden Tests stehen für iOS-Geräte zur Verfügung.

- Automatische Appium-Tests
- XCTest

- XCTest Benutzeroberfläche

Integrierte Testtypen für iOS

Es ist derzeit nur eine integrierte Testart für iOS-Geräte verfügbar.

- Eingebaut: Fuzz (Android und iOS)

Integrieren von Device Farm mit XCTest für iOS

Mit Device Farm können Sie das XCTest Framework verwenden, um Ihre App auf echten Geräten zu testen. Weitere Informationen dazu finden Sie XCTest unter Grundlagen des Testens beim Testen mit Xcode.

Um einen Test auszuführen, erstellen Sie die Pakete für Ihren Testlauf und laden diese Pakete auf Device Farm hoch.

Weitere Informationen zum Testen in Device Farm finden Sie unterTest-Frameworks und integrierte Tests in AWS Device Farm.

Themen

- Erstellen Sie die Pakete für Ihren XCTest Lauf
- Laden Sie die Pakete für Ihren XCTest Lauf auf Device Farm hoch

Erstellen Sie die Pakete für Ihren XCTest Lauf

Um Ihre App mithilfe des XCTest Frameworks zu testen, benötigt Device Farm Folgendes:

- Ihr App-Paket verfügt über eine `.ipa`-Datei.
- Ihr XCTest Paket als `.zip` Datei.

Sie erstellen diese Pakete mithilfe der von Xcode generierten Build-Ausgabe. Gehen Sie wie folgt vor, um die Pakete zu erstellen, damit Sie sie auf Device Farm hochladen können.

So generieren Sie die Build-Ausgabe für Ihre App:

1. Öffnen Sie Ihr App-Projekt in Xcode.

2. Wählen Sie im Schema-Dropdownmenü auf der Xcode-Toolleiste Generisches iOS-Gerät als Ziel.
3. Wählen Sie im Menü Product (Produkt) Build For (Build für) und dann Testing (Test).

So erstellen Sie das App-Paket:

1. Öffnen Sie im Projektnavigator in Xcode unter Products (Produkte) das Kontextmenü für die Datei mit dem Namen *app-project-name*`.app`. Wählen Sie dann Show in Finder (im Finder anzeigen). Der Finder öffnet einen Ordner mit dem Namen `Debug-iphoneos`, der die Ausgabe enthält, die Xcode für Ihren test-Build generiert hat. Dieser Ordner enthält Ihre `.app`-Datei.
2. Erstellen Sie im Finder einen neuen Ordner, und benennen Sie ihn `Payload`.
3. Kopieren Sie die *app-project-name*`.app`-Datei, und fügen Sie sie in den Ordner `Payload` ein.
4. Öffnen Sie das Kontextmenü für den Ordner `Payload` und wählen Sie Compress „Payload“ („Payload“ komprimieren). Eine Datei mit dem Namen `Payload.zip` wird erstellt.
5. Ändern Sie den Namen der Datei und ihre Erweiterung `Payload.zip` zu *app-project-name*`.ipa`.

   In einem späteren Schritt stellen Sie diese Datei Device Farm zur Verfügung. Um die Datei leichter auffindbar zu machen, sollten Sie sie zu einem anderen ort verschieben, etwa auf Ihr Desktop.
6. Optional können Sie den Ordner `Payload` und die Datei `.app` darin löschen.

Um das XCTest Paket zu erstellen

1. Öffnen Sie im Finder im Verzeichnis `Debug-iphoneos` das Kontextmenü für die Datei *app-project-name*`.app`. Wählen Sie dann Show Package Contents (Paketinhalte anzeigen).
2. Öffnen Sie in den Paketinhalten den Ordner `Plugins`. Dieser Ordner enthält eine Datei mit dem Namen *app-project-name*`.xctest`.
3. Öffnen Sie das Kontextmenü für diese Datei und wählen Sie „Komprimieren“ ***app-project-name*`.xctest`**. Eine Datei mit dem Namen *app-project-name*`.xctest.zip` wird erstellt.

   In einem späteren Schritt stellen Sie diese Datei Device Farm zur Verfügung. Um die Datei leichter auffindbar zu machen, sollten Sie sie zu einem anderen ort verschieben, etwa auf Ihr Desktop.

Laden Sie die Pakete für Ihren XCTest Lauf auf Device Farm hoch

Verwenden Sie die Device Farm Farm-Konsole, um die Pakete für Ihren Test hochzuladen.

1. Melden Sie sich bei der Device Farm Farm-Konsole unter https://console.aws.amazon.com/devicefarm an.
2. Wenn Sie noch kein Projekt haben, erstellen Sie eines. Die Schritte zum Erstellen eines Projekts finden Sie unter Ein Projekt in AWS Device Farm erstellen.

   Andernfalls wählen Sie im Navigationsbereich der Device Farm die Option Mobile Device Testing und dann Projects aus.
3. Wählen Sie das Projekt aus, mit dem Sie den Test ausführen möchten.
4. Wählen Sie Testlauf erstellen aus.
5. Wählen Sie unter Ausführungseinstellungen im Abschnitt Ausführungstyp die Option iOS-App aus.
6. Wählen Sie unter App auswählen im Abschnitt App-Auswahloptionen die Option Eigene App hochladen aus. Wählen Sie dann unter App hochladen die Option Datei auswählen aus.
7. Navigieren Sie zur `.ipa`-Datei für Ihre App und laden Sie sie hoch.

   > **Note**
   >
   > Ihr `.ipa`-Paket muss für Tests erstellt sein.
8. Wählen Sie unter Test konfigurieren im Abschnitt Testframework auswählen die Option XCTest. Wählen Sie dann unter App hochladen die Option Datei auswählen aus.
9. Suchen Sie nach der `.zip` Datei, die das XCTest Paket für Ihre App enthält, und laden Sie es hoch.
10. Führen Sie die restlichen Schritte der Projekterstellung durch. Sie wählen die Geräte für den Test und geben den gerätezustand an.
11. Wählen Sie Create run aus. Device Farm führt Ihren Test aus und zeigt die Ergebnisse in der Konsole an.

Integrieren der XCTest Benutzeroberfläche für iOS mit Device Farm

Device Farm bietet Unterstützung für das XCTest UI-Testframework. Insbesondere unterstützt Device Farm XCTest UI-Tests, die sowohl in Objective-C als auch in Swift geschrieben wurden.

Das XCTest UI-Framework ermöglicht UI-Tests in der iOS-Entwicklung, aufbauend auf XCTest. Weitere Informationen finden Sie unter User Interface Testing in der iOS Developer Library.

Allgemeine Informationen zum Testen in Device Farm finden Sie unterTest-Frameworks und integrierte Tests in AWS Device Farm.

Verwenden Sie die folgenden Anweisungen, um Device Farm in das XCTest UI-Testframework für iOS zu integrieren.

Themen

Bereiten Sie Ihre XCTest iOS-UI-Tests vor

Sie können entweder eine `.ipa` Datei oder eine `.zip` Datei für Ihr XCTEST_UI-Testpaket hochladen.

Eine `.ipa` Datei ist ein Anwendungsarchiv, das die iOS Runner-App im Bundle-Format enthält. Zusätzliche Dateien können nicht in die *`.ipa`* Datei aufgenommen werden.

Wenn Sie eine `.zip` Datei hochladen, kann sie entweder direkt die iOS Runner-App oder eine `.ipa` Datei enthalten. Sie können der `.zip` Datei auch andere Dateien hinzufügen, wenn Sie sie während der Tests verwenden möchten. Sie können beispielsweise Dateien wie `.xcworkspace` oder `.xcodeproj` in eine `.zip` Datei einfügen`.xctestrun`, um XCUI-Testpläne auf der Gerätefarm auszuführen. Detaillierte Anweisungen zur Ausführung von Testplänen finden Sie in der Standardtestspezifikationsdatei für den XCUI-Testtyp.

Option 1: Erstellen eines XCTest UI-Packages (.ipa)

Das Bundle yourAppNameUITest-Runner.app wird von Xcode erstellt, wenn Sie Ihr Projekt zum Testen erstellen. Es ist im Verzeichnis "Products" für das Projekt zu finden.

Um eine IPA-Datei zu erstellen:

1. Erstellen Sie ein Verzeichnis mit dem Namen. *`Payload`*
2. Fügen Sie Ihr App-Verzeichnis dem Payload-Verzeichnis hinzu.

3. Archivieren Sie das Payload-Verzeichnis in einer `.zip` Datei und ändern Sie dann die Dateierweiterung in. `.ipa`

Die folgende Ordnerstruktur zeigt, wie eine Beispiel-App mit dem Namen als `.ipa` Datei verpackt werden *`my-project-nameUITest-Runner.app`* würde:

```
.
### my-project-nameUITest.ipa
    ### Payload (directory)
        ### my-project-nameUITest-Runner.app
```

Option 2: Erstellen eines XCTest UI-Pakets im ZIP-Format

Device Farm generiert automatisch eine `.xctestrun` Datei für Sie, mit der Sie Ihre vollständige XCTest UI-Testsuite ausführen können. Wenn Sie Ihre eigene `.xctestrun` Datei auf Device Farm verwenden möchten, können Sie Ihre `.xctestrun` Dateien und das App-Verzeichnis in eine `.zip` Datei komprimieren. Wenn Sie bereits eine `.ipa` Datei für Ihr Testpaket haben, können Sie diese stattdessen hier einfügen*`*-Runner.app`*.

```
.
### swift-sample-UI.zip (directory)
   ### my-project-nameUITest-Runner.app [OR] my-project-nameUITest.ipa
   ### SampleTestPlan_2.xctestrun
   ### SampleTestPlan_1.xctestrun
   ### (any other files)
```

Wenn Sie einen Xcode-Testplan für Ihre XCUI-Tests auf Device Farm ausführen möchten, können Sie eine ZIP-Datei erstellen, die Ihre my-project-nameUITest-Runner.app - oder my-project-name UITest .ipa-Datei und die Xcode-Quellcodedateien enthält, die für die Ausführung von XCTEST_UI mit Testplänen erforderlich sind, einschließlich einer OR-Datei. `.xcworkspace` `.xcodeproj`

Hier `.xcodeproj` ist ein Beispiel für eine ZIP-Datei, die eine Datei verwendet:

```
.
### swift-sample-UI.zip (directory)
   ### my-project-nameUITest-Runner.app [OR] my-project-nameUITest.ipa
   ### (any directory)
```

```
   ### SampleXcodeProject.xcodeproj
        ### Testplan_1.xctestplan
        ### Testplan_2.xctestplan
        ### (any other source code files created by xcode with .xcodeproj)
```

Hier ist ein Beispiel für eine Zip-Datei, die eine `.xcworkspace` Datei verwendet:

```
.
###swift-sample-UI.zip (directory)
   ### my-project-nameUITest-Runner.app [OR] my-project-nameUITest.ipa
   ### (any directory)
   #   ### SampleXcodeProject.xcodeproj
   #   ### Testplan_1.xctestplan
   #   ### Testplan_2.xctestplan
   |   ### (any other source code files created by xcode with .xcodeproj)
   ### SampleWorkspace.xcworkspace
       ### contents.xcworkspacedata
```

> **Note**
>
> Bitte stellen Sie sicher, dass Ihr XCTest UI−.zip-Paket kein Verzeichnis mit dem Namen „Payload“ enthält.

Laden Sie Ihre XCTest iOS-UI-Tests hoch

Verwenden Sie die Device Farm Farm-Konsole, um Ihre Tests hochzuladen.

1. Melden Sie sich bei der Device Farm Farm-Konsole unter https://console.aws.amazon.com/devicefarm an.
2. Wählen Sie im Navigationsbereich Device Farm die Option Mobile Device Testing und dann Projects aus.
3. Wählen Sie in der Projektliste das Projekt aus, in das Sie Ihre Tests hochladen möchten.

   > **Tip**
   >
   > Sie können die Suchleiste verwenden, um die Projektliste nach Namen zu filtern.

Um ein Projekt zu erstellen, folgen Sie den Anweisungen unter Ein Projekt in AWS Device Farm erstellen

4. Wählen Sie Lauf erstellen.
5. Wählen Sie unter Ausführungseinstellungen im Abschnitt Ausführungstyp die Option iOS-App aus.
6. Wählen Sie unter App auswählen im Abschnitt App-Auswahloptionen die Option Eigene App hochladen aus. Wählen Sie dann unter App hochladen die Option Datei auswählen aus.
7. Navigieren Sie zu der Datei mit Ihrer iOS-Anwendung und wählen Sie diese aus. Es muss sich dabei um eine IPA-Datei handeln.

> **Note**
>
> Stellen Sie sicher, dass Ihre IPA-Datei für ein iOS-Gerät und nicht für einen Simulator erstellt wurde.

8. Wählen Sie unter Test konfigurieren im Abschnitt Testframework auswählen die Option XCTest UI aus. Wählen Sie dann unter App hochladen die Option Datei auswählen aus.
9. Suchen Sie die IPA- oder ZIP-Datei, die Ihren XCTest iOS-UI-Test-Runner enthält, und wählen Sie sie aus.
10. Schließen Sie die verbleibenden Schritte bei der Erstellung des Laufs ab. Sie wählen die Geräte aus, auf denen Sie testen möchten, und geben optional eine zusätzliche Konfiguration an.
11. Wählen Sie Create Run aus. Device Farm führt Ihren Test aus und zeigt die Ergebnisse in der Konsole an.

Web-App-Tests in AWS Device Farm

Device Farm bietet Tests mit Appium für Webanwendungen. Weitere Informationen zum Einrichten Ihrer Appium-Tests auf Device Farm finden Sie unterthe section called “Automatische Appium-Tests”.

Weitere Informationen zum Testen in Device Farm finden Sie unterTest-Frameworks und integrierte Tests in AWS Device Farm.

Regeln für Geräte mit und ohne Messgerät

Der Begriff der Zählerüberwachung bezieht sich auf die Abrechnung pro Gerät. Standardmäßig werden Device Farm Farm-Geräte gemessen und Ihnen wird pro Minute berechnet, nachdem

die kostenlosen Testminuten aufgebraucht sind. Sie können auch nicht zählerüberwachte Geräte erwerben und für eine feste monatliche Gebühr unbegrenzt Tests durchführen. Weitere Informationen zur Preisgestaltung finden Sie unter Preise für AWS Device Farm.

Wenn Sie einen Lauf für einen Geräte-Pool starten möchten, in dem sowohl iOS als auch Android-Geräte enthalten Sind, werden bestimmte Regeln für zählerüberwachte und nicht zählerüberwachte Geräte angewendet. Wenn Sie z. B. über fünf nicht zählerüberwachte Android-Geräte und fünf nicht zählerüberwachte iOS-Geräte verfügen, werden für Ihre Web-Testlauf Ihre nicht zählerüberwachte Geräte verwendet.

Ein anderes Beispiel: Nehmen wir an, Sie verfügen über fünf nicht zählerüberwachte Android-Geräte und 0 nicht zählerüberwachte iOS-Geräte. Wenn Sie nur die Android-Geräte für den Web-Testlauf auswählen, werden Ihre nicht zählerüberwachten Geräte verwendet. Wenn Sie sowohl Android- als auch iOS-Geräte für den Web-Testlauf auswählen, wird die zählerüberwachte Abrechnungsmethode angewendet, und Ihre nicht zählerüberwachten Geräte werden nicht verwendet.

Integrierte Tests in AWS Device Farm

Device Farm bietet Unterstützung für integrierte Testtypen für Android- und iOS-Geräte.

Mit integrierten Tests können Sie Ihre Anwendung auf mehreren Geräten testen, ohne Testautomatisierungsskripts schreiben und verwalten zu müssen. Dies kann Ihnen Zeit und Mühe sparen, insbesondere wenn Sie mit Device Farm beginnen. Device Farm bietet den folgenden integrierten Testtyp:

- Eingebaut: Fuzz (Android und iOS)— Der Fuzz-Test sendet zufällig Benutzeroberflächenereignisse an Geräte und meldet dann Ergebnisse.

Weitere Informationen zu Tests und Testframeworks in Device Farm finden Sie unterTest-Frameworks und integrierte Tests in AWS Device Farm.

Ausführen des integrierten Fuzz-Tests von Device Farm (Android und iOS)

Der integrierte Fuzz-Test von Device Farm sendet nach dem Zufallsprinzip Benutzeroberflächenereignisse an Geräte und meldet dann Ergebnisse.

Weitere Informationen zum Testen in Device Farm finden Sie unterTest-Frameworks und integrierte Tests in AWS Device Farm.

So führen Sie den integrierten Fuzz-Test aus

1. Melden Sie sich bei der Device Farm Farm-Konsole unter https://console.aws.amazon.com/devicefarm an.
2. Wählen Sie im Navigationsbereich Device Farm die Option Mobile Device Testing und dann Projects aus.
3. Wählen Sie in der Projektliste das Projekt aus, in dem Sie den integrierten Fuzz-Test ausführen möchten.

   > **Tip**
   >
   > Sie können die Suchleiste verwenden, um die Projektliste nach Namen zu filtern. Befolgen Sie die Anweisungen unter Ein Projekt in AWS Device Farm erstellen, um ein neues Projekt zu erstellen.

4. Wählen Sie Lauf erstellen.
5. Wählen Sie unter Laufeinstellungen im Abschnitt Ausführungstyp Ihren Ausführungstyp aus. Wählen Sie Android-App aus, wenn Sie noch keine App zum Testen bereit haben oder wenn Sie eine Android-App (.apk) testen. Wählen Sie iOS-App, wenn Sie eine iOS-App (.ipa) testen.
6. Wählen Sie unter App auswählen die Option Von Device Farm bereitgestellte Beispiel-App auswählen aus, wenn Sie keine App zum Testen zur Verfügung haben. Wenn Sie Ihre eigene App mitbringen, wählen Sie Eigene App hochladen und wählen Sie Ihre Anwendungsdatei aus.
7. Wählen Sie unter Test konfigurieren im Abschnitt Testframework auswählen die Option Integriert: Fuzz aus.
8. Wenn eine oder mehrere der folgenden Einstellungen angezeigt wird, können Sie entweder die Standardwerte übernehmen oder eigene Werte angeben:

   - Event count (Ereignisanzahl): Geben Sie eine Zahl zwischen 1 und 10.000 ein, welche die Anzahl der Benutzeroberflächen-Ereignisse angibt, die für den auszuführenden Fuzz-Test generiert werden soll.
   - Event-Throttling: Geben Sie eine Zahl zwischen 0 und 1.000 an, die angibt, wie viele Millisekunden der Fuzz-Test warten muss, bevor das nächste Benutzeroberflächenereignis ausgeführt wird.
   - Randomizer seed (Randomisierungs-Seed): Geben Sie eine Zahl an, die vom auszuführenden Fuzz-Test als Startwert für die Randomisierung von Benutzeroberflächen-Ereignissen

verwendet werden soll. Die Verwendung desselben Werts für aufeinander folgende Fuzz-Tests gewährleistet identische Ereignissequenzen.

9. Folgen Sie den verbleibenden Anweisungen, um Geräte auszuwählen und den Rechenlauf zu starten.

Benutzerdefinierte Testumgebungen in AWS Device Farm

AWS Device Farm ermöglicht die Konfiguration einer benutzerdefinierten Umgebung für automatisierte Tests (benutzerdefinierter Modus). Dies ist der empfohlene Ansatz für alle Device Farm Farm-Benutzer. Weitere Informationen zu Umgebungen in Device Farm finden Sie unter Testumgebungen.

Der benutzerdefinierte Modus bietet im Vergleich zum Standardmodus unter anderem folgende Vorteile:

- Schnellere end-to-end Testausführung: Das Testpaket wird nicht analysiert, um jeden Test in der Suite zu erkennen, wodurch preprocessing/postprocessing Overhead vermieden wird.
- Live-Protokoll und Videostreaming: Ihre clientseitigen Testprotokolle und Videos werden live gestreamt, wenn Sie den benutzerdefinierten Modus verwenden. Diese Funktion ist im Standardmodus nicht verfügbar.
- Erfasst alle Artefakte: Auf dem Host und dem Gerät können Sie im benutzerdefinierten Modus alle Testartefakte erfassen. Dies ist im Standardmodus möglicherweise nicht möglich.
- Konsistentere und replizierbarere lokale Umgebung: Im Standardmodus werden Artefakte für jeden einzelnen Test separat bereitgestellt, was unter bestimmten Umständen von Vorteil sein kann. Ihre lokale Testumgebung kann jedoch von der ursprünglichen Konfiguration abweichen, da Device Farm jeden ausgeführten Test unterschiedlich behandelt.

  Im Gegensatz dazu können Sie im benutzerdefinierten Modus Ihre Device Farm Farm-Testausführungsumgebung konsistent an Ihre lokale Testumgebung anpassen.

Benutzerdefinierte Umgebungen werden mithilfe einer Testspezifikationsdatei (Testspezifikation) im YAML-Format konfiguriert. Device Farm stellt für jeden unterstützten Testtyp eine Standardtestspezifikationsdatei bereit, die unverändert oder angepasst verwendet werden kann. Anpassungen wie Testfilter oder Konfigurationsdateien können der Testspezifikation hinzugefügt werden. Bearbeitete Testspezifikationen können für future Testläufe gespeichert werden.

Weitere Informationen finden Sie unter Hochladen einer benutzerdefinierten Testspezifikation mit dem AWS CLI und. Einen Testlauf in Device Farm erstellen

Themen

- Referenz und Syntax der Testspezifikation

- Hosts für benutzerdefinierte Testumgebungen
- Greifen Sie mithilfe einer IAM-Ausführungsrolle auf AWS-Ressourcen zu
- Umgebungsvariablen für benutzerdefinierte Testumgebungen
- Bewährte Methoden für die Ausführung benutzerdefinierter Testumgebungen
- Migrieren von Tests von einer Standard- zu einer benutzerdefinierten Testumgebung
- Erweiterung benutzerdefinierter Testumgebungen in Device Farm

Referenz und Syntax der Testspezifikation

Die Testspezifikation (Testspezifikation) ist eine Datei, mit der Sie benutzerdefinierte Testumgebungen in Device Farm definieren.

Arbeitsablauf für Testspezifikationen

Die Device Farm Farm-Testspezifikation führt Phasen und ihre Befehle in einer vordefinierten Reihenfolge aus, sodass Sie die Art und Weise, wie Ihre Umgebung vorbereitet und ausgeführt wird, anpassen können. Wenn jede Phase ausgeführt wird, werden ihre Befehle in der Reihenfolge ausgeführt, die in der Testspezifikationsdatei aufgeführt ist. Die Phasen werden in der folgenden Reihenfolge ausgeführt

1. `install`- Hier sollten Aktionen wie das Herunterladen, Installieren und Einrichten von Tools definiert werden.
2. `pre_test`- Hier sollten Aktionen vor dem Test definiert werden, z. B. das Starten von Hintergrundprozessen.
3. `test`- Hier sollte der Befehl definiert werden, der Ihren Test aufruft.
4. `post_test`- Hier sollten alle letzten Aufgaben definiert werden, die nach Abschluss Ihres Tests ausgeführt werden müssen, z. B. die Generierung von Testberichten und die Aggregation von Artefaktdateien.

Syntax der Testspezifikationen

Das Folgende ist das YAML-Schema für eine Testspezifikationsdatei

```
version: 0.1
```

```
android_test_host: "string"
ios_test_host: "string"

phases:
  install:
    commands:
      - "string"
      - "string"
  pre_test:
    commands:
      - "string"
      - "string"
  test:
    commands:
      - "string"
      - "string"
  post_test:
    commands:
      - "string"
      - "string"

artifacts:
    - "string"
    - "string"
```

`version`

(Erforderlich, Nummer)

Spiegelt die von Device Farm unterstützte Version der Testspezifikation wider. Die aktuelle Versionsnummer lautet `0.1`.

`android_test_host`

(Optional, Zeichenfolge)

Der Testhost, der für Testläufe auf Android-Geräten ausgewählt wird. Dieses Feld ist für Testläufe auf Android-Geräten erforderlich. Weitere Informationen finden Sie unter Verfügbare Testhosts für benutzerdefinierte Testumgebungen.

`ios_test_host`

(Optional, Zeichenfolge)

Der Testhost, der für Testläufe auf iOS-Geräten ausgewählt wird. Dieses Feld ist für Testläufe auf iOS-Geräten mit einer Hauptversion größer als 26 erforderlich. Weitere Informationen finden Sie unter Verfügbare Testhosts für benutzerdefinierte Testumgebungen.

`phases`

Dieser Abschnitt enthält Gruppen von Befehlen, die während eines Testlaufs ausgeführt werden, wobei jede Phase optional ist. Die zulässigen Namen der Testphasen `install` lauten:`pre_test`,`test`, und`post_test`.

- `install`- Standardabhängigkeiten für Test-Frameworks, die von Device Farm unterstützt werden, sind bereits installiert. Diese Phase enthält gegebenenfalls zusätzliche Befehle, die Device Farm während der Installation ausführt.
- `pre_test`- Die Befehle, falls vorhanden, wurden vor dem automatisierten Test ausgeführt.
- `test`- Die Befehle, die während Ihres automatisierten Testlaufs ausgeführt wurden. Wenn ein Befehl in der Testphase fehlschlägt (was bedeutet, dass er einen Exit-Code ungleich Null zurückgibt), wird der Test als fehlgeschlagen markiert
- `post_test`- Die Befehle, falls vorhanden, werden nach Ihrem automatisierten Testlauf ausgeführt. Dies wird unabhängig davon ausgeführt, ob Ihr Test in der `test` Phase erfolgreich ist oder nicht.

`commands`

(Optional, Liste [Zeichenfolge])

Eine Liste von Zeichenketten, die während der Phase als Shell-Befehl ausgeführt werden sollen.

`artifacts`

(Optional, Liste [Zeichenfolge])

Device Farm sammelt Artefakte wie benutzerdefinierte Berichte, Protokolldateien und Bilder von einem hier angegebenen Speicherort. Platzhalterzeichen werden als Teil eines Artefakt-Speicherortes nicht unterstützt. Sie müssen einen gültigen Pfad für jeden Speicherort eingeben.

Diese Testartefakte sind für jedes Gerät in Ihrem Testlauf verfügbar. Weitere Informationen zum Abrufen Ihrer Testartefakte finden Sie unter Artefakte werden in einer benutzerdefinierten Testumgebung heruntergeladen.

> **Important**
>
> Eine Testspezifikation muss als gültige YAML-Datei formatiert werden. Wenn die Einrückungen oder Leerstellen in Ihrer Testspezifikation ungültig sind, kann Ihr Testlauf fehlschlagen. Registerkarten sind in YAML-Dateien nicht zulässig. Sie können mit einem YAML-Validator testen, ob Ihre Testspezifikation eine gültige YAML-Datei ist. Weitere Informationen finden Sie auf der YAML-Website.

Beispiele für Testspezifikationen

Die folgenden Beispiele zeigen Testspezifikationen, die auf Device Farm ausgeführt werden können.

Simple Demo

Im Folgenden finden Sie ein Beispiel für eine Testspezifikationsdatei, die einfach `Hello world!` als Testlaufartefakt protokolliert wird.

```
version: 0.1

android_test_host: amazon_linux_2
ios_test_host: macos_sequoia

phases:
  install:
    commands:
      # Setup your environment by installing and/or validating software
      - devicefarm-cli use python 3.11
      - python --version

  pre_test:
    commands:
      # Setup your tests by starting background tasks or setting up
      # additional environment variables.
      - OUTPUT_FILE="/tmp/hello.log"

  test:
    commands:
      # Run your tests within this phase.
      - python -c 'print("Hello world!")' &> $OUTPUT_FILE

  post_test:
```

```
    commands:
      # Perform any remaining tasks within this phase, such as copying
      # artifacts to the DEVICEFARM_LOG_DIR for upload
      - cp $OUTPUT_FILE $DEVICEFARM_LOG_DIR

artifacts:
  # By default, Device Farm will collect your artifacts from the $DEVICEFARM_LOG_DIR
 directory.
  - $DEVICEFARM_LOG_DIR
```

Appium Android

Im Folgenden finden Sie ein Beispiel für eine Testspezifikationsdatei, mit der ein Appium Java TestNG-Testlauf auf Android konfiguriert wird.

```
version: 0.1

# The following fields(s) allow you to select which Device Farm test host is used
 for your test run.
android_test_host: amazon_linux_2

phases:

  # The install phase contains commands for installing dependencies to run your
 tests.
  # Certain frequently used dependencies are preinstalled on the test host to
 accelerate and
  # simplify your test setup. To find these dependencies, versions supported and
 additional
  # software installation please see:
  # https://docs.aws.amazon.com/devicefarm/latest/developerguide/custom-test-
environments-hosts-software.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 20
      - 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 periodically 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@2.19.0

      # When running Android tests with Appium version 2, the uiautomator2 driver is
preinstalled using driver
      # version 2.44.1 for Appium 2.5.1  If you want to install a different 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;

      # Based on Appium framework's recommendation, we recommend setting the Appium
server's
      # base path explicitly for accepting commands. If you prefer the legacy base
path of /wd/hub,
      # please set it here.
      - export APPIUM_BASE_PATH=

      # Use the devicefarm-cli to setup a Java environment, with which you can run
your test suite.
      - devicefarm-cli use java 17
      - java -version

 # The pre-test phase contains commands for setting up your test environment.
 pre_test:
   commands:
     # Setup the CLASSPATH so that Java knows where to find your test classes.
     - export CLASSPATH=$CLASSPATH:$DEVICEFARM_TEST_PACKAGE_PATH/*
     - export CLASSPATH=$CLASSPATH:$DEVICEFARM_TEST_PACKAGE_PATH/dependency-jars/*

     # 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: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 &;

      # This code snippet is to 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.
      - echo "Navigate to test package directory"
      - cd $DEVICEFARM_TEST_PACKAGE_PATH
      - echo "Starting the Appium TestNG test"

      # The following command runs your Appium Java TestNG test.
```

```
      # For more information, please see TestNG's documentation here:
      # https://testng.org/#_running_testng
      - |-
        java -Dappium.screenshots.dir=$DEVICEFARM_SCREENSHOT_PATH org.testng.TestNG
 -testjar *-tests.jar \
          -d $DEVICEFARM_LOG_DIR/test-output -verbose 10

      # To run your tests with a testng.xml file that is a part of your test
 package,
      # use the following commands instead:

      # - echo "Unzipping the tests JAR file"
      # - unzip *-tests.jar
      # - |-
      #   java -Dappium.screenshots.dir=$DEVICEFARM_SCREENSHOT_PATH
 org.testng.TestNG -testjar *-tests.jar \
      #     testng.xml -d $DEVICEFARM_LOG_DIR/test-output -verbose 10

  # 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, with certain limits
  (see limit details
# here: https://docs.aws.amazon.com/devicefarm/latest/developerguide/
limits.html#file-limits).
# 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
```

Appium iOS

Im Folgenden finden Sie ein Beispiel für eine Testspezifikationsdatei, mit der ein Appium Java TestNG-Testlauf unter iOS konfiguriert wird.

```
version: 0.1

# The following fields(s) allow you to select which Device Farm test host is used
 for your test run.
ios_test_host: macos_sequoia

phases:

  # The install phase contains commands for installing dependencies to run your
 tests.
  # Certain frequently used dependencies are preinstalled on the test host to
 accelerate and
  # simplify your test setup. To find these dependencies, versions supported and
 additional
  # software installation please see:
  # https://docs.aws.amazon.com/devicefarm/latest/developerguide/custom-test-
environments-hosts-software.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 20
      - 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 periodically 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@2.19.0

      # When running iOS tests with Appium version 2, the XCUITest driver is
 preinstalled using driver
      # version 9.10.5 for Appium 2.5.4. 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@10.0.0;
      #   fi;

      # Based on Appium framework's recommendation, we recommend setting the Appium
server's
      # base path explicitly for accepting commands. If you prefer the legacy base
path of /wd/hub,
      # please set it here.
      - export APPIUM_BASE_PATH=

      # Use the devicefarm-cli to setup a Java environment, with which you can run
your test suite.
      - devicefarm-cli use java 17
      - java -version

 # The pre-test phase contains commands for setting up your test environment.
 pre_test:
   commands:
     # Setup the CLASSPATH so that Java knows where to find your test classes.
     - export CLASSPATH=$CLASSPATH:$DEVICEFARM_TEST_PACKAGE_PATH/*
     - export CLASSPATH=$CLASSPATH:$DEVICEFARM_TEST_PACKAGE_PATH/dependency-jars/*

      # Device Farm provides multiple pre-built versions of WebDriverAgent (WDA), a
required
      # Appium dependency for iOS, where each version corresponds to the XCUITest
driver version selected.
      # If Device Farm cannot find a corresponding version of WDA for your XCUITest
driver,
      # the latest available version is selected by default.
      - |-
        APPIUM_DRIVER_VERSION=$(appium driver list --installed --json | jq -r
".xcuitest.version" | cut -d "." -f 1);
        CORRESPONDING_APPIUM_WDA=$(env | grep
"DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V${APPIUM_DRIVER_VERSION}")
        if [[ ! -z "$APPIUM_DRIVER_VERSION" ]] && [[ ! -z
"$CORRESPONDING_APPIUM_WDA" ]]; then
          echo "Using Device Farm's prebuilt WDA version ${APPIUM_DRIVER_VERSION}.x,
which corresponds with your driver";
```

```
          DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH=$(echo $CORRESPONDING_APPIUM_WDA |
 cut -d "=" -f2)
        else
          LATEST_SUPPORTED_WDA_VERSION=$(env | grep
 "DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V" | sort -V -r | head -n 1)
          echo "Unknown driver version $APPIUM_DRIVER_VERSION; falling back to the
 Device Farm default version of $LATEST_SUPPORTED_WDA_VERSION";
          DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH=$(echo
 $LATEST_SUPPORTED_WDA_VERSION | cut -d "=" -f2)
        fi;

      # For iOS versions 16 and below only, the device unique identifier (UDID)
 needs to modified for Appium tests
      # on Device Farm to remove the hypens.
      - |-
        if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "iOS" ]; then
          DEVICEFARM_DEVICE_UDID_FOR_APPIUM=$DEVICEFARM_DEVICE_UDID;
          if [ $(echo $DEVICEFARM_DEVICE_OS_VERSION | cut -d "." -f 1) -le 16 ];
 then
            DEVICEFARM_DEVICE_UDID_FOR_APPIUM=$(echo $DEVICEFARM_DEVICE_UDID | tr -d
 "-");
          fi;
        fi;

      # 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_FOR_APPIUM\", \
          \"appium:platformVersion\": \"$DEVICEFARM_DEVICE_OS_VERSION\", \
          \"appium:derivedDataPath\": \"$DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH\",
 \
```

```
          \"appium:usePrebuiltWDA\": true, \
          \"appium:automationName\": \"XCUITest\"}" \
          >> $DEVICEFARM_LOG_DIR/appium.log 2>&1 &

     # This code snippet is to 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.
     - echo "Navigate to test package directory"
     - cd $DEVICEFARM_TEST_PACKAGE_PATH
     - echo "Starting the Appium TestNG test"

     # The following command runs your Appium Java TestNG test.
     # For more information, please see TestNG's documentation here:
     # https://testng.org/#_running_testng
     - |-
       java -Dappium.screenshots.dir=$DEVICEFARM_SCREENSHOT_PATH org.testng.TestNG
-testjar *-tests.jar \
         -d $DEVICEFARM_LOG_DIR/test-output -verbose 10

     # To run your tests with a testng.xml file that is a part of your test
package,
     # use the following commands instead:

     # - echo "Unzipping the tests JAR file"
     # - unzip *-tests.jar
     # - |-
     #   java -Dappium.screenshots.dir=$DEVICEFARM_SCREENSHOT_PATH
org.testng.TestNG -testjar *-tests.jar \
```

```
      #     testng.xml -d $DEVICEFARM_LOG_DIR/test-output -verbose 10

  # 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, with certain limits
 (see limit details
# here: https://docs.aws.amazon.com/devicefarm/latest/developerguide/
limits.html#file-limits).
# 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
```

Appium (Both Platforms)

Im Folgenden finden Sie ein Beispiel für eine Testspezifikationsdatei, mit der ein Appium Java TestNG-Testlauf auf Android und iOS konfiguriert wird.

```
version: 0.1

# The following fields(s) allow you to select which Device Farm test host is used
 for your test run.
android_test_host: amazon_linux_2
ios_test_host: macos_sequoia

phases:

  # The install phase contains commands for installing dependencies to run your
 tests.
  # Certain frequently used dependencies are preinstalled on the test host to
 accelerate and
  # simplify your test setup. To find these dependencies, versions supported and
 additional
```

```
  # software installation please see:
  # https://docs.aws.amazon.com/devicefarm/latest/developerguide/custom-test-
environments-hosts-software.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 20
      - 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 periodically 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@2.19.0

      # When running Android tests with Appium version 2, the uiautomator2 driver is
 preinstalled using driver
      # version 2.44.1 for Appium 2.5.1  If you want to install a different 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;

      # When running iOS tests with Appium version 2, the XCUITest driver is
 preinstalled using driver
      # version 9.10.5 for Appium 2.5.4. 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@10.0.0;
     #   fi;

     # Based on Appium framework's recommendation, we recommend setting the Appium
server's
     # base path explicitly for accepting commands. If you prefer the legacy base
path of /wd/hub,
     # please set it here.
     - export APPIUM_BASE_PATH=

     # Use the devicefarm-cli to setup a Java environment, with which you can run
your test suite.
     - devicefarm-cli use java 17
     - java -version

 # The pre-test phase contains commands for setting up your test environment.
 pre_test:
   commands:
     # Setup the CLASSPATH so that Java knows where to find your test classes.
     - export CLASSPATH=$CLASSPATH:$DEVICEFARM_TEST_PACKAGE_PATH/*
     - export CLASSPATH=$CLASSPATH:$DEVICEFARM_TEST_PACKAGE_PATH/dependency-jars/*

     # Device Farm provides multiple pre-built versions of WebDriverAgent (WDA), a
required
     # Appium dependency for iOS, where each version corresponds to the XCUITest
driver version selected.
     # If Device Farm cannot find a corresponding version of WDA for your XCUITest
driver,
     # the latest available version is selected by default.
     - |-
       if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "iOS" ]; then
         APPIUM_DRIVER_VERSION=$(appium driver list --installed --json | jq -r
".xcuitest.version" | cut -d "." -f 1);
         CORRESPONDING_APPIUM_WDA=$(env | grep
"DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V${APPIUM_DRIVER_VERSION}")
         if [[ ! -z "$APPIUM_DRIVER_VERSION" ]] && [[ ! -z
"$CORRESPONDING_APPIUM_WDA" ]]; then
           echo "Using Device Farm's prebuilt WDA version
${APPIUM_DRIVER_VERSION}.x, which corresponds with your driver";
```

```
            DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH=$(echo $CORRESPONDING_APPIUM_WDA
 | cut -d "=" -f2)
          else
            LATEST_SUPPORTED_WDA_VERSION=$(env | grep
 "DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V" | sort -V -r | head -n 1)
            echo "Unknown driver version $APPIUM_DRIVER_VERSION; falling back to the
 Device Farm default version of $LATEST_SUPPORTED_WDA_VERSION";
            DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH=$(echo
 $LATEST_SUPPORTED_WDA_VERSION | cut -d "=" -f2)
          fi;
        fi;

      # For iOS versions 16 and below only, the device unique identifier (UDID)
 needs to modified for Appium tests
      # on Device Farm to remove the hypens.
      - |-
        if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "iOS" ]; then
          DEVICEFARM_DEVICE_UDID_FOR_APPIUM=$DEVICEFARM_DEVICE_UDID;
          if [ $(echo $DEVICEFARM_DEVICE_OS_VERSION | cut -d "." -f 1) -le 16 ];
 then
            DEVICEFARM_DEVICE_UDID_FOR_APPIUM=$(echo $DEVICEFARM_DEVICE_UDID | tr -d
 "-");
          fi;
        fi;

      # 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: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: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 snippet is to 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.
      - echo "Navigate to test package directory"
      - cd $DEVICEFARM_TEST_PACKAGE_PATH
      - echo "Starting the Appium TestNG test"

      # The following command runs your Appium Java TestNG test.
      # For more information, please see TestNG's documentation here:
      # https://testng.org/#_running_testng
      - |-
```

```
        java -Dappium.screenshots.dir=$DEVICEFARM_SCREENSHOT_PATH org.testng.TestNG
 -testjar *-tests.jar \
          -d $DEVICEFARM_LOG_DIR/test-output -verbose 10

     # To run your tests with a testng.xml file that is a part of your test
 package,
     # use the following commands instead:

     # - echo "Unzipping the tests JAR file"
     # - unzip *-tests.jar
     # - |-
     #   java -Dappium.screenshots.dir=$DEVICEFARM_SCREENSHOT_PATH
 org.testng.TestNG -testjar *-tests.jar \
     #     testng.xml -d $DEVICEFARM_LOG_DIR/test-output -verbose 10

  # 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, with certain limits
 (see limit details
# here: https://docs.aws.amazon.com/devicefarm/latest/developerguide/
limits.html#file-limits).
# 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
```

Hosts für benutzerdefinierte Testumgebungen

Device Farm unterstützt eine Reihe von Betriebssystemen mit vorkonfigurierter Software mithilfe einer Testhostumgebung. Während der Testausführung verwendet Device Farm von Amazon verwaltete Instanzen (Hosts), die sich dynamisch mit dem ausgewählten Testgerät verbinden.

Diese Instanz wird vollständig bereinigt und zwischen den Testläufen nicht wiederverwendet. Nach Abschluss des Testlaufs wird sie mit den generierten Artefakten beendet.

Themen

Verfügbare Testhosts für benutzerdefinierte Testumgebungen

Die Testhosts werden vollständig von Device Farm verwaltet. In der folgenden Tabelle sind die derzeit verfügbaren und unterstützten Device Farm Farm-Testhosts für benutzerdefinierte Testumgebungen aufgeführt.

Geräteplattform	Host testen	Betriebssystem	Architektur (en)	Unterstützte Geräte
Android	amazon_linux_2	Amazon Linux 2	x86_64	Android6 und höher
iOS	macos_sequoia	macOS Sequoia(Version 15)	arm64	iOS15 bis 26

> **Note**
>
> In regelmäßigen Abständen fügt Device Farm neue Testhosts für eine Geräteplattform hinzu, um neuere Gerätebetriebssystemversionen und deren Abhängigkeiten zu unterstützen. In diesem Fall läuft der Support für ältere Testhosts für die jeweilige Geräteplattform ab.

Version des Betriebssystems

Jeder verfügbare Testhost verwendet eine bestimmte Version des Betriebssystems, das zu diesem Zeitpunkt auf Device Farm unterstützt wird. Obwohl wir versuchen, die neueste Betriebssystemversion zu verwenden, ist dies möglicherweise nicht die neueste öffentlich verfügbare Version. Device Farm aktualisiert das Betriebssystem regelmäßig mit kleineren Versionsupdates und Sicherheitspatches.

Um die spezifische Version (einschließlich der Nebenversion) des Betriebssystems zu ermitteln, die während des Testlaufs verwendet wurde, können Sie den folgenden Codeausschnitt zu jeder Phase Ihrer Testspezifikationsdatei hinzufügen.

Example

```
phases:
  install:
    commands:
      # The following example prints the instance's operating system version details
      - |-
        if [[ "Darwin" == "$(uname)" ]]; then
          echo "$(sw_vers --productName) $(sw_vers --productVersion) ($(sw_vers --
buildVersion))";
        else
          echo "$(. /etc/os-release && echo $PRETTY_NAME) ($(uname -r))";
        fi
```

Auswahl eines Testhosts für benutzerdefinierte Testumgebungen

Sie können den Android- und iOS-Testhost in den entsprechenden `ios_test_host` Variablen `android_test_host` und Ihrer Testspezifikationsdatei angeben.

Wenn Sie keine Testhostauswahl für die angegebene Geräteplattform angeben, werden Tests auf dem Testhost ausgeführt, den Device Farm als Standard für das angegebene Gerät und die angegebene Testkonfiguration festgelegt hat.

> **Important**
>
> Beim Testen auf iOS 18 und niedriger wird ein älterer Testhost verwendet, wenn kein Host ausgewählt ist. Weitere Informationen finden Sie im Thema auf derLegacy-iOS-Testhost.

Sehen Sie sich als Beispiel den folgenden Codeausschnitt an:

Example

```
version: 0.1
android_test_host: amazon_linux_2
ios_test_host: macos_sequoia

phases:
  # ...
```

Unterstützte Software in benutzerdefinierten Testumgebungen

Device Farm verwendet Host-Computer, auf denen viele der erforderlichen Softwarebibliotheken vorinstalliert sind, um Test-Frameworks auszuführen, die von unserem Service unterstützt werden, und bietet beim Start eine einsatzbereite Testumgebung. Device Farm unterstützt mithilfe unseres Softwareauswahlmechanismus mehrere Sprachen und aktualisiert regelmäßig die Versionen der in der Umgebung enthaltenen Sprachen.

Für jede andere erforderliche Software können Sie die Testspezifikationsdatei so ändern, dass sie von Ihrem Testpaket aus installiert, aus dem Internet heruntergeladen oder auf private Quellen in Ihrer VPC zugegriffen wird (weitere Informationen finden Sie unter VPC ENI). Weitere Informationen finden Sie unter Beispiele für Testspezifikationen.

Vorkonfigurierte Software

Um das Testen von Geräten auf jeder Plattform zu erleichtern, stehen auf dem Testhost die folgenden Tools zur Verfügung:

Tools	Geräteplattform (en)
Android SDK Build-Tools	Android
Android SDK Platform-Tools(beinhaltetadb)	Android
Xcode	iOS

Wählbare Software

Zusätzlich zur vorkonfigurierten Software auf dem Host bietet Device Farm die Möglichkeit, bestimmte Versionen unterstützter Software über das `devicefarm-cli` Tooling auszuwählen.

Die folgende Tabelle enthält die auswählbare Software und die Testhosts, die sie enthalten.

Software/Tool	Hosts, die diese Software unterstützen	Befehl zur Verwendung in Ihrer Testspezifikation
Java 17	amazon_linux_2 macos_sequoia	`devicefarm-cli use java 17`
Java 11	amazon_linux_2 macos_sequoia	`devicefarm-cli use java 11`
Java 8	amazon_linux_2 macos_sequoia	`devicefarm-cli use java 8`
Node.js 20	amazon_linux_2 macos_sequoia	`devicefarm-cli use node 20`
Node.js 18	amazon_linux_2 macos_sequoia	`devicefarm-cli use node 18`
Node.js 16	amazon_linux_2	`devicefarm-cli use node 16`
Python 3.11	amazon_linux_2 macos_sequoia	`devicefarm-cli use python 3.11`
Python 3.10	amazon_linux_2 macos_sequoia	`devicefarm-cli use python 3.10`

Software/Tool	Hosts, die diese Software unterstützen	Befehl zur Verwendung in Ihrer Testspezifikation
Python 3.9	amazon_linux_2 macos_sequoia	`devicefarm-cli use python 3.9`
Python 3.8	amazon_linux_2	`devicefarm-cli use python 3.8`
Ruby 3.2	amazon_linux_2 macos_sequoia	`devicefarm-cli use ruby 3.2`
Ruby 2.7	amazon_linux_2	`devicefarm-cli use ruby 2.7`
Appium 3	amazon_linux_2	`devicefarm-cli use appium 3`
Appium 2	amazon_linux_2 macos_sequoia	`devicefarm-cli use appium 2`
Appium 1	amazon_linux_2	`devicefarm-cli use appium 1`
Xcode 26	macos_sequoia	`devicefarm-cli use xcode 26`
Xcode 16	macos_sequoia	`devicefarm-cli use xcode 16`

Der Testhost enthält auch häufig verwendete Unterstützungstools für jede Softwareversion, wie z. B. die `npm` Paketmanager `pip` und die Paketmanager (jeweils in Python und Node.js enthalten) und Abhängigkeiten (wie den UIAutomator2 Appium-Treiber) für Tools wie Appium. Dadurch wird sichergestellt, dass Sie über die Tools verfügen, die Sie für die Arbeit mit den unterstützten Test-Frameworks benötigen.

Verwenden des Devicefarm-cli-Tools in benutzerdefinierten Testumgebungen

Der Testhost verwendet ein standardisiertes Versionsverwaltungstool, das `devicefarm-cli` zur Auswahl von Softwareversionen aufgerufen wird. Dieses Tool ist unabhängig vom Device Farm-Testhost AWS CLI und nur auf dem Device Farm Farm-Testhost verfügbar. Mit `devicefarm-cli` können Sie zu einer beliebigen vorinstallierten Softwareversion auf dem Testhost wechseln. Dies bietet eine einfache Möglichkeit, Ihre Device Farm Farm-Testspezifikationsdatei im Laufe der Zeit zu verwalten, und bietet Ihnen einen vorhersehbaren Mechanismus, um Softwareversionen in future zu aktualisieren.

> **Important**
>
> Dieses Befehlszeilentool ist auf älteren iOS-Hosts nicht verfügbar. Weitere Informationen finden Sie im Thema auf derLegacy-iOS-Testhost.

Der folgende Ausschnitt zeigt die `help` Seite von: `devicefarm-cli`

```
$ 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.
```

Sehen wir uns einige Beispiele mit an. `devicefarm-cli` Führen Sie die folgenden Befehle aus, um das Tool *`3.10`* zum Ändern der Python-Version von zu *`3.9`* in Ihrer Testspezifikationsdatei zu verwenden:

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

So ändern Sie die Appium-Version von *`1`* zu: *`2`*

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

> **Tip**
>
> Beachten Sie, dass bei der Auswahl einer Softwareversion `devicefarm-cli` auch die unterstützenden Tools für diese Sprachen, z. B. `pip` für Python und NodeJS, `npm` gewechselt werden.

Weitere Informationen zur vorinstallierten Software auf dem Testhost finden Sie unter. Unterstützte Software in benutzerdefinierten Testumgebungen

Testumgebung für Android-Geräte

AWS Device Farm verwendet Amazon Elastic Compute Cloud (EC2) -Hostmaschinen, auf denen Amazon Linux 2 ausgeführt wird, um Android-Tests auszuführen. Wenn Sie einen Testlauf planen, weist Device Farm jedem Gerät einen eigenen Host zu, damit die Tests unabhängig voneinander ausgeführt werden können. Die Hostcomputer werden nach dem Testlauf zusammen mit allen generierten Artefakten beendet.

Der Amazon Linux 2-Host bietet mehrere Vorteile:

- Schnelleres, zuverlässigeres Testen: Im Vergleich zum alten Host verbessert der neue Testhost die Testgeschwindigkeit erheblich und reduziert insbesondere die Teststartzeiten. Der Amazon Linux 2-Host weist auch beim Testen eine höhere Stabilität und Zuverlässigkeit auf.
- Verbesserter Fernzugriff für manuelle Tests: Upgrades auf den neuesten Testhost und Verbesserungen führen zu einer geringeren Latenz und einer besseren Videoleistung für manuelle Android-Tests.
- Auswahl der Standard-Softwareversion: Device Farm standardisiert jetzt die Unterstützung wichtiger Programmiersprachen auf dem Testhost sowie die Appium-Framework-Versionen. Für unterstützte Sprachen (derzeit Java, Python, Node.js und Ruby) und Appium bietet der neue Testhost bald nach dem Start langfristige stabile Releases. Die zentralisierte Versionsverwaltung über das `devicefarm-cli` Tool ermöglicht die Entwicklung von Testspezifikationsdateien mit einer konsistenten Erfahrung in allen Frameworks.

Themen

Unterstützte IP-Bereiche für die Amazon Linux 2-Testumgebung in Device Farm

Kunden müssen häufig den IP-Bereich kennen, aus dem der Traffic von Device Farm stammt, insbesondere für die Konfiguration ihrer Firewalls und Sicherheitseinstellungen. Bei Amazon EC2-Testhosts umfasst der IP-Bereich die gesamte `us-west-2` Region. Für Amazon Linux 2-Testhosts, die Standardoption für neue Android-Läufe, wurden die Bereiche eingeschränkt. Der Datenverkehr stammt jetzt von einer bestimmten Gruppe von NAT-Gateways, wodurch der IP-Bereich auf die folgenden Adressen beschränkt ist:

IP-Bereiche
44.236.137.143
52,13,151,244
522,35,189,191
54,201,250,26

Weitere Informationen zu Android-Testumgebungen in Device Farm finden Sie unterTestumgebung für Android-Geräte.

Testumgebung für iOS-Geräte

Device Farm verwendet von Amazon verwaltete macOS-Instanzen (Hosts), die sich während des Testlaufs dynamisch mit dem iOS-Gerät verbinden. Jeder Host ist mit Software vorkonfiguriert, die Gerätetests auf verschiedenen gängigen Testplattformen wie XCTest UI und Appium ermöglicht.

Die aktuelle Version des iOS-Testhosts hat das Testerlebnis im Vergleich zu früheren Versionen verbessert, darunter:

- Konsistente Host-Betriebssystem- und Tooling-Erfahrung für iOS 15 bis iOS 26 Bisher wurde der Testhost vom verwendeten Gerät bestimmt, was bei der Ausführung auf mehreren iOS-Versionen zu einer fragmentierten Softwareumgebung führte. Die derzeitige Erfahrung ermöglicht eine einfache Hostauswahl, um eine konsistente Umgebung auf allen Geräten zu gewährleisten.

Dadurch können dieselbe macOS-Version und dieselben Tools (wie Xcode) auf jedem iOS-Gerät verfügbar sein.

- Leistungsverbesserungen für iOS 15- und 16-Tests Mithilfe der aktualisierten Infrastruktur hat sich die Einrichtungszeit für iOS 15- und 16-Tests erheblich verbessert.
- Standardisierte auswählbare Softwareversionen für unterstützte Abhängigkeiten Wir haben jetzt das `devicefarm-cli` Softwareauswahlsystem sowohl auf iOS- als auch auf Android-Testhosts, sodass Sie Ihre bevorzugte Version unserer unterstützten Abhängigkeiten auswählen können. Für unterstützte Abhängigkeiten (wie Java, Python, Node.js, Ruby und Appium) können Versionen über die Testspezifikation ausgewählt werden. Eine Vorstellung davon, wie diese Funktion funktioniert, finden Sie im Thema unter. Unterstützte Software in benutzerdefinierten Testumgebungen

> **Important**
>
> Wenn Sie auf iOS 18 und niedriger ausgeführt werden, werden Ihre Tests standardmäßig auf älteren Testhosts ausgeführt. Im folgenden Thema erfahren Sie, wie Sie von Legacy-Hosts weg migrieren können.

Legacy-iOS-Testhost

Für bestehende Tests unter iOS 18 und niedriger werden die Legacy-Testhosts standardmäßig für benutzerdefinierte Testumgebungen ausgewählt. Die folgende Tabelle enthält die Test-Host-Version, mit der die iOS-Geräteversion ausgeführt wird.

Betriebssystem	Architektur (en)	Standard für Geräte
macOS Sonoma(Version 14)	arm64	iOS 18
macOS Ventura(Ausführung 13)	arm64	iOS 17
macOS Monterey(Ausführung 12)	x86_64	iOS 16und darunter

Informationen zur Auswahl der neueren Testhosts finden Sie im entsprechenden ThemaMigrieren Sie Ihre benutzerdefinierten Testumgebungen auf die neuen iOS-Testhosts.

Unterstützte Software für iOS-Geräte

Um iOS-Gerätetests zu unterstützen, sind Device Farm Farm-Testhosts für iOS-Geräte mit Xcode und den zugehörigen Befehlszeilentools vorkonfiguriert. Weitere verfügbare Software finden Sie im folgenden Thema. Unterstützte Software in benutzerdefinierten Testumgebungen

Migrieren Sie Ihre benutzerdefinierten Testumgebungen auf die neuen iOS-Testhosts

Um bestehende Tests vom Legacy-Host auf den neuen macOS-Testhost zu migrieren, müssen Sie neue Testspezifikationsdateien entwickeln, die auf Ihren bereits vorhandenen basieren.

Der empfohlene Ansatz besteht darin, mit der Beispieltestspezifikationsdatei für die gewünschten Testtypen zu beginnen und dann relevante Befehle von Ihrer alten Testspezifikationsdatei auf die neue zu migrieren. Auf diese Weise können Sie neue Funktionen und Optimierungen der Beispieltestspezifikation für den neuen Host nutzen und gleichzeitig Ausschnitte aus Ihrem vorhandenen Code wiederverwenden.

Themen

- Tutorial: Migrieren von iOS-Testspezifikationsdateien mit der Konsole
- Unterschiede zwischen den neuen und den älteren Testhosts

Tutorial: Migrieren von iOS-Testspezifikationsdateien mit der Konsole

In diesem Beispiel wird die Device Farm Farm-Konsole verwendet, um eine bestehende iOS-Gerätetestspezifikation zu integrieren, um den neuen Testhost zu verwenden.

Schritt 1: Erstellen neuer Testspezifikationsdateien mit der Konsole

1. Melden Sie sich bei der AWS Device Farm Farm-Konsole an.
2. Navigieren Sie zum Device Farm Farm-Projekt, das Ihre Automatisierungstests enthält.
3. Laden Sie eine Kopie der vorhandenen Testspezifikation herunter, die Sie verwenden möchten.
    a. Klicken Sie auf die Option „Projekteinstellungen“ und navigieren Sie zur Registerkarte Uploads.
    b. Navigieren Sie zu der Testspezifikationsdatei, die Sie verwenden möchten.
    c. Klicken Sie auf die Schaltfläche Herunterladen, um eine lokale Kopie dieser Datei zu erstellen.

4. Gehen Sie zurück zur Projektseite und klicken Sie auf Ausführung erstellen.
5. Füllen Sie die Optionen im Assistenten so aus, als ob Sie einen neuen Lauf starten würden, halten Sie aber bei der Option Testspezifikation auswählen an.
6. Verwenden Sie die standardmäßig ausgewählte iOS-Testspezifikation und klicken Sie auf die Schaltfläche Testspezifikation erstellen.
7. Ändern Sie die Testspezifikation, die standardmäßig im Texteditor ausgewählt wurde.

    a. Falls noch nicht vorhanden, ändern Sie die Testspezifikationsdatei, um den neuen Host auszuwählen. Verwenden Sie dazu:

    ```
    ios_test_host: macos_sequoia
    ```

    b. Überprüfen Sie jede Kopie Ihrer Testspezifikation, die Sie in einem vorherigen Schritt heruntergeladen haben. `phase`

    c. Kopiert Befehle aus den Phasen der alten Testspezifikation in die jeweiligen Phasen der neuen Testspezifikation und ignoriert dabei Befehle, die sich auf die Installation oder Auswahl von Java, Python, Node.js, Ruby, Appium oder Xcode beziehen.
8. Geben Sie einen neuen Dateinamen in das Textfeld Speichern unter ein.
9. Klicken Sie auf die Schaltfläche Als neu speichern, um Ihre Änderungen zu speichern.

Ein Beispiel für eine Testspezifikationsdatei, die Sie als Referenz verwenden können, finden Sie im Beispiele für Testspezifikationen Beispiel unter.

Schritt 2: Auswahl der vorinstallierten Software

Auf dem neuen Testhost werden vorinstallierte Softwareversionen mithilfe eines neuen standardisierten Versionsverwaltungstools namens ausgewählt. `devicefarm-cli` Dieses Tool ist jetzt der empfohlene Ansatz für die Verwendung der verschiedenen Software, die wir auf den Testhosts bereitstellen.

Als Beispiel würden Sie die folgende Zeile hinzufügen, um ein anderes JDK 17 in Ihrer Testumgebung zu verwenden:

```
- devicefarm-cli use java 17
```

Weitere Informationen zur unterstützten und verfügbaren Software finden Sie unter:Unterstützte Software in benutzerdefinierten Testumgebungen.

Schritt 3: Verwenden von Appium und seinen Abhängigkeiten über das Softwareauswahltool

Der neue Testhost unterstützt nur Appium 2.x und höher. Bitte wählen Sie explizit die Appium-Version mit dem `devicefarm-cli` aus und entfernen Sie dabei ältere Tools wie. `avm` Beispiel:

```
# This line using 'avm' should be removed
# - avm 2.3.1

# And the following lines should be added
- devicefarm-cli use appium 2 # Selects the version
- appium --version            # Prints the version
```

Die mit ausgewählte Appium-Version `devicefarm-cli` ist mit einer kompatiblen Version des XCUITest Treibers für iOS vorinstalliert.

Darüber hinaus müssen Sie Ihre Testspezifikation aktualisieren, um sie anstelle von zu verwenden `DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V9`. `DEVICEFARM_WDA_DERIVED_DATA_PATH` Die neue Umgebungsvariable verweist auf eine vorgefertigte Version von WebDriverAgent 9.x, der neuesten unterstützten Version für Appium 2-Tests.

Weitere Informationen finden Sie unter und. Auswahl einer WebDriverAgent Version für iOS-Tests Umgebungsvariablen für Appium-Tests

Unterschiede zwischen den neuen und den älteren Testhosts

Wenn Sie Ihre Testspezifikationsdatei bearbeiten, um den neuen iOS-Testhost zu verwenden, und Ihre Tests vom alten Testhost umstellen, sollten Sie sich der folgenden wichtigen Umgebungsunterschiede bewusst sein:

- Xcode-Versionen: In der Legacy-Testhostumgebung basierte die verfügbare Xcode-Version auf der iOS-Version des zum Testen verwendeten Geräts. Beispielsweise verwendeten Tests auf iOS 18-Geräten Xcode 16 im Legacy-Host, wohingegen Tests auf iOS 17 Xcode 15 verwendeten. In der neuen Host-Umgebung können alle Geräte auf dieselben Versionen von Xcode zugreifen, was eine konsistente Umgebung für Tests auf Geräten mit unterschiedlichen Versionen ermöglicht. Eine Liste der derzeit verfügbaren Xcode-Versionen finden Sie unter. Unterstützte Software
- Softwareversionen auswählen: In vielen Fällen haben sich die Standard-Softwareversionen geändert. Wenn Sie also Ihre Softwareversion auf dem älteren Testhost nicht explizit ausgewählt haben, möchten Sie sie jetzt vielleicht auf dem neuen Testhost mit `devicefarm-cli`angeben. In den allermeisten Anwendungsfällen empfehlen wir Kunden, explizit die Softwareversionen

auszuwählen, die sie verwenden. Wenn Sie eine Softwareversion mit auswählen, haben `devicefarm-cli` Sie eine vorhersehbare und konsistente Erfahrung damit und erhalten zahlreiche Warnungen, wenn Device Farm plant, diese Version vom Testhost zu entfernen.

Darüber hinaus `rvm` wurden Tools zur Softwareauswahl wie `nvm` `pyenv` `avm`,, und zugunsten des neuen `devicefarm-cli` Softwareauswahlsystems entfernt.

- Verfügbare Softwareversionen: Viele Versionen zuvor vorinstallierter Software wurden entfernt und viele neue Versionen hinzugefügt. Stellen Sie daher sicher, dass Sie bei der `devicefarm-cli` Auswahl Ihrer Softwareversionen Versionen auswählen, die in der Liste der unterstützten Versionen enthalten sind.
- Die **`libimobiledevice`** Toolsuite wurde zugunsten neuerer Tools bzw. Tools von Erstanbietern entfernt, um aktuelle iOS-Gerätetests und Industriestandards nachzuverfolgen. Für iOS 17 und höher können Sie die meisten Befehle migrieren, um ähnliche Xcode-Tools zu verwenden, genannt. `devicectl` Informationen dazu finden Sie unter: Sie können das Programm `xcrun devicectl help` von einem Computer aus ausführen`devicectl`, auf dem Xcode installiert ist.
- Dateipfade, die in Ihrer alten Host-Testspezifikationsdatei als absolute Pfade fest codiert sind, funktionieren auf dem neuen Testhost höchstwahrscheinlich nicht wie erwartet und werden im Allgemeinen nicht für die Verwendung von Testspezifikationsdateien empfohlen. Wir empfehlen, relative Pfade und Umgebungsvariablen für den gesamten Code der Testspezifikationsdatei zu verwenden. Weitere Informationen finden Sie im Thema unterBewährte Methoden für die Ausführung benutzerdefinierter Testumgebungen.
- Betriebssystemversion und Architektur: Die älteren Testhosts verwendeten eine Vielzahl von macOS-Versionen und CPU-Architekturen, die auf dem zugewiesenen Gerät basierten. Daher stellen Benutzer möglicherweise einige Unterschiede in den verfügbaren Systembibliotheken fest, die in der Umgebung verfügbar sind. Weitere Informationen zur vorherigen Host-Betriebssystemversion finden Sie unterLegacy-iOS-Testhost.
- Für Appium-Benutzer WebDriverAgent wurde die Art der Auswahl von dahingehend geändert, dass `DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V` anstelle des alten `DEVICEFARM_WDA_DERIVED_DATA_PATH_V` Präfixes ein Präfix für die Verwendung einer Umgebungsvariablen verwendet wird. Weitere Informationen zur aktualisierten Variablen finden Sie Umgebungsvariablen für Appium-Tests unter.
- Für Appium Java-Benutzer enthält der neue Testhost keine vorinstallierten JAR-Dateien in seinem Klassenpfad, wohingegen der vorherige Host eine für das TestNG-Framework (über eine Umgebungsvariable) enthielt. `$DEVICEFARM_TESTNG_JAR` Wir empfehlen Kunden, die erforderlichen JAR-Dateien für ihre Testframeworks in ihr Testpaket zu packen und Instanzen der `$DEVICEFARM_TESTNG_JAR` Variablen aus ihren Testspezifikationsdateien zu entfernen.

Wir empfehlen, sich über einen Support-Fall an das Serviceteam zu wenden, wenn Sie Feedback oder Fragen zu den Unterschieden zwischen den Testhosts aus Sicht der Software haben.

Greifen Sie mithilfe einer IAM-Ausführungsrolle auf AWS-Ressourcen zu

Device Farm unterstützt die Angabe einer IAM-Rolle, die während der Testausführung von der benutzerdefinierten Testlaufzeitumgebung übernommen wird. Mit dieser Funktion können Ihre Tests sicher auf AWS-Ressourcen in Ihrem Konto zugreifen, z. B. Amazon S3 S3-Buckets, DynamoDB-Tabellen oder andere AWS-Services, von denen Ihre Anwendung abhängt.

Themen

Übersicht

Wenn Sie eine IAM-Ausführungsrolle angeben, übernimmt Device Farm diese Rolle während der Testausführung, sodass Ihre Tests mithilfe der in der Rolle definierten Berechtigungen mit AWS-Services interagieren können.

Zu den häufigsten Anwendungsfällen für IAM-Ausführungsrollen gehören:

- Zugreifen auf Testdaten, die in Amazon S3 S3-Buckets gespeichert sind
- Testartefakte in Amazon S3 S3-Buckets übertragen
- Anwendungskonfiguration von AWS abrufen AppConfig
- Schreiben von Testprotokollen und Metriken auf Amazon CloudWatch
- Senden von Testergebnissen oder Statusmeldungen an Amazon SQS SQS-Warteschlangen
- Aufrufen von AWS Lambda Lambda-Funktionen als Teil von Test-Workflows

IAM-Rollenanforderungen

Um eine IAM-Ausführungsrolle mit Device Farm verwenden zu können, muss Ihre Rolle die folgenden Anforderungen erfüllen:

- Vertrauensverhältnis: Dem Device Farm Farm-Dienstprinzipal muss vertraut werden, um die Rolle übernehmen zu können. Die Vertrauensrichtlinie muss eine `devicefarm.amazonaws.com` vertrauenswürdige Entität beinhalten.
- Berechtigungen: Die Rolle muss über die erforderlichen Berechtigungen für den Zugriff auf die AWS-Ressourcen verfügen, die für Ihre Tests erforderlich sind.
- Sitzungsdauer: Die maximale Sitzungsdauer der Rolle muss mindestens so lang sein wie die Job-Timeout-Einstellung Ihres Device Farm Farm-Projekts. Standardmäßig haben Device Farm Farm-Projekte ein Job-Timeout von 150 Minuten, sodass Ihre Rolle eine Sitzungsdauer von mindestens 150 Minuten unterstützen muss.
- Gleiche Kontoanforderung: Die IAM-Rolle muss sich in demselben AWS-Konto befinden wie das, mit dem Device Farm aufgerufen wurde. Die kontoübergreifende Übernahme von Rollen wird nicht unterstützt.
- PassRole Berechtigung: Der Aufrufer muss gemäß einer Richtlinie autorisiert sein, die IAM-Rolle zu übergeben, sodass die `iam:PassRole` Aktion für die angegebene Ausführungsrolle zulässig ist.

Beispiel für eine Treuhand-Police

Das folgende Beispiel zeigt eine Vertrauensrichtlinie, die es Device Farm ermöglicht, Ihre Ausführungsrolle zu übernehmen. Diese Vertrauensrichtlinie sollte nur der spezifischen IAM-Rolle zugewiesen werden, die Sie mit Device Farm verwenden möchten, nicht anderen Rollen in Ihrem Konto:

Example

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "devicefarm.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Beispiel für eine Berechtigungsrichtlinie

Das folgende Beispiel zeigt eine Berechtigungsrichtlinie, die Zugriff auf allgemeine AWS-Services gewährt, die beim Testen verwendet werden:

Example

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::my-test-bucket",
        "arn:aws:s3:::my-test-bucket/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "appconfig:GetConfiguration",
        "appconfig:StartConfigurationSession"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:*:*:log-group:/devicefarm/test-*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "sqs:SendMessage",
        "sqs:GetQueueUrl"
      ],
      "Resource": "arn:aws:sqs:*:*:test-results-*"
    }
  ]
}
```

Konfiguration einer IAM-Ausführungsrolle

Sie können eine IAM-Ausführungsrolle auf Projektebene oder für einzelne Testläufe angeben. Bei der Konfiguration auf Projektebene erben alle Läufe innerhalb dieses Projekts die Ausführungsrolle. Eine bei einem Lauf konfigurierte Ausführungsrolle ersetzt alle in ihrem übergeordneten Projekt konfigurierten Rollen.

Detaillierte Anweisungen zur Konfiguration von Ausführungsrollen finden Sie unter:

- Ein Projekt in AWS Device Farm erstellen- zur Konfiguration von Ausführungsrollen auf Projektebene
- Einen Testlauf in Device Farm erstellen- zur Konfiguration von Ausführungsrollen für einzelne Läufe

Sie können Ausführungsrollen auch mithilfe der Device Farm API konfigurieren. Weitere Informationen finden Sie in der Device Farm API-Referenz.

Best Practices

Folgen Sie diesen bewährten Methoden, wenn Sie IAM-Ausführungsrollen für Ihre Device Farm Farm-Tests konfigurieren:

- Prinzip der geringsten Rechte: Gewähren Sie nur die Mindestberechtigungen, die für das Funktionieren Ihrer Tests erforderlich sind. Vermeiden Sie die Verwendung zu weit gefasster Berechtigungen wie * Aktionen oder Ressourcen.
- Verwenden Sie ressourcenspezifische Berechtigungen: Beschränken Sie die Berechtigungen nach Möglichkeit auf bestimmte Ressourcen (z. B. bestimmte S3-Buckets oder DynamoDB-Tabellen) und nicht auf alle Ressourcen eines Typs.
- Separate Test- und Produktionsressourcen: Verwenden Sie spezielle Testressourcen und Rollen, um zu verhindern, dass während des Tests versehentlich Produktionssysteme beeinträchtigt werden.
- Regelmäßige Rollenüberprüfung: Überprüfen und aktualisieren Sie Ihre Ausführungsrollen regelmäßig, um sicherzustellen, dass sie weiterhin Ihren Testanforderungen entsprechen und den bewährten Sicherheitsmethoden entsprechen.
- Verwenden Sie Bedingungsschlüssel: Erwägen Sie die Verwendung von IAM-Bedingungsschlüsseln, um weiter einzuschränken, wann und wie die Rolle verwendet werden kann.

Fehlerbehebung

Wenn Sie Probleme mit IAM-Ausführungsrollen haben, überprüfen Sie Folgendes:

- Vertrauensverhältnis: Stellen Sie sicher, dass die Vertrauensrichtlinie der Rolle einen vertrauenswürdigen Dienst beinhaltet`devicefarm.amazonaws.com`.
- Berechtigungen: Stellen Sie sicher, dass die Rolle über die erforderlichen Berechtigungen für die AWS-Services verfügt, auf die Ihre Tests zugreifen möchten.
- Testprotokolle: Überprüfen Sie die Testausführungsprotokolle auf spezifische Fehlermeldungen im Zusammenhang mit AWS-API-Aufrufen oder Verweigerungen von Berechtigungen.

Umgebungsvariablen für benutzerdefinierte Testumgebungen

Device Farm konfiguriert dynamisch mehrere Umgebungsvariablen für die Verwendung als Teil Ihrer benutzerdefinierten Testumgebung.

Themen

Benutzerdefinierte Umgebungsvariablen

Device Farm unterstützt die Konfiguration von Schlüssel-Wert-Paaren, die als Umgebungsvariablen auf dem Testhost angewendet werden. Diese können in einem Device Farm Farm-Projekt oder während der Lauferstellung konfiguriert werden. Alle Variablen, die bei einem Lauf konfiguriert wurden, ersetzen alle Variablen, die für das übergeordnete Projekt konfiguriert wurden. Beachten Sie die folgenden Einschränkungen:

- Benutzerdefinierte Umgebungsvariablen werden auf älteren iOS-Testhosts nicht unterstützt. Weitere Informationen finden Sie unter Legacy-iOS-Testhost.
- Variablennamen, die mit beginnen, `$DEVICEFARM_` sind für die interne Verwendung durch Dienste reserviert.

- Benutzerdefinierte Umgebungsvariablen dürfen nicht verwendet werden, um die Computerauswahl für den Testhost in Ihrer Testspezifikation zu konfigurieren.

Allgemeine Umgebungsvariablen

In diesem Abschnitt werden Umgebungsvariablen beschrieben, die allen Tests in Device Farm gemeinsam sind.

`$DEVICEFARM_DEVICE_NAME`

Das Gerät, auf dem Ihre Tests ausgeführt werden. Es stellt die eindeutige Gerätekennung (UDID) des Geräts dar.

`$DEVICEFARM_DEVICE_UDID`

Die eindeutige Kennung des Geräts.

`$DEVICEFARM_DEVICE_PLATFORM_NAME`

Der Plattformname des Geräts. Es ist entweder `Android` oder`iOS`.

`$DEVICEFARM_DEVICE_OS_VERSION`

Die Betriebssystemversion des Geräts.

`$DEVICEFARM_APP_PATH`

(Tests für mobile Apps)

Der Pfad zur mobilen App auf dem Host-Computer, auf dem die Tests ausgeführt werden. Diese Variable ist bei Webtests nicht verfügbar.

`$DEVICEFARM_LOG_DIR`

Der Pfad zum Standardverzeichnis, in dem Kundenprotokolle, Artefakte und andere benötigte Dateien für den späteren Abruf gespeichert werden. Anhand einer Beispieltestspezifikation werden Dateien in diesem Verzeichnis in einer ZIP-Datei archiviert und nach dem Testlauf als Artefakt zur Verfügung gestellt.

`$DEVICEFARM_SCREENSHOT_PATH`

Der Pfad zu den Screenshots, die ggf. während des Testlaufs erfasst werden.

`$DEVICEFARM_PROJECT_ARN`

Der ARN des übergeordneten Projekts des Jobs.

$DEVICEFARM_RUN_ARN

Der ARN der übergeordneten Ausführung des Jobs.

$DEVICEFARM_DEVICE_ARN

Der ARN des zu testenden Geräts.

$DEVICEFARM_TOTAL_JOBS

Die Gesamtzahl der Jobs, die mit der Ausführung der übergeordneten Device Farm verknüpft sind.

$DEVICEFARM_JOB_NUMBER

Die Nummer dieses Jobs ist darin enthalten`$DEVICEFARM_TOTAL_JOBS`. Ein Lauf kann beispielsweise 5 Jobs enthalten, von denen jeder einen eindeutigen Wert `$DEVICEFARM_JOB_NUMBER` zwischen 0 und 4 hat.

$AWS_REGION

Die AWS-Region Der Dienst stellt dies so ein, dass es der Region entspricht, in der sich das zu testende Gerät befindet. Sie kann bei Bedarf durch eine benutzerdefinierte Umgebungsvariable überschrieben werden.

$ANDROID_HOME

(Nur Android)

Der Pfad zum Android SDK-Installationsverzeichnis.

Umgebungsvariablen für Appium-Tests

In diesem Abschnitt werden Umgebungsvariablen beschrieben, die von jedem Appium-Test in einer benutzerdefinierten Testumgebung in Device Farm verwendet werden.

$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR

(Nur Android)

Der Speicherort eines Verzeichnisses, das die erforderlichen ChromeDriver ausführbaren Dateien für die Verwendung in Appium-Web- und Hybridtests enthält.

$DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V<N>

(nur iOS)

Der abgeleitete Datenpfad einer Version von, die für die Ausführung auf Device Farm WebDriverAgent erstellt wurde. Die Nummerierung der Variablen entspricht der Hauptversion von. WebDriverAgent Als Beispiel `DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V9` wird auf die WebDriverAgent A-Version von 9.x verwiesen. Weitere Informationen finden Sie unter Auswahl einer WebDriverAgent Version für iOS-Tests.

> **Note**
>
> Die `$DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V<N>` Umgebungsvariablen sind nur auf Nicht-Legacy-iOS-Hosts vorhanden. Weitere Informationen finden Sie unter Legacy-iOS-Testhost.

$DEVICEFARM_WDA_DERIVED_DATA_PATH_V9

(Nur iOS, veraltet)

Der abgeleitete Datenpfad einer Version von, die für die Ausführung auf Device Farm WebDriverAgent erstellt wurde. Das Ersatzbenennungsschema finden Sie unter. `$DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V<N>`

Umgebungsvariablen für XCUITest Tests

In diesem Abschnitt werden Umgebungsvariablen beschrieben, die vom XCUITest Test in einer benutzerdefinierten Testumgebung in Device Farm verwendet werden.

$DEVICEFARM_XCUITESTRUN_FILE

Der Pfad zur Device Farm `.xctestun` Farm-Datei. Wird über Ihre App und aus Testpaketen erstellt.

$DEVICEFARM_DERIVED_DATA_PATH

Erwarteter Pfad der Xcodebuild-Ausgabe von Device Farm.

$DEVICEFARM_XCTEST_BUILD_DIRECTORY

Der Pfad zum entpackten Inhalt der Test-Paketdatei.

Bewährte Methoden für die Ausführung benutzerdefinierter Testumgebungen

In den folgenden Themen werden empfohlene bewährte Methoden für die Verwendung der benutzerdefinierten Testausführung mit Device Farm behandelt.

Ausführungskonfiguration

- Verlassen Sie sich bei der Ausführungskonfiguration wo immer möglich auf von Device Farm verwaltete Software und API-Funktionen, anstatt ähnliche Konfigurationen über Shell-Befehle in der Testspezifikationsdatei anzuwenden. Dazu gehört auch die Konfiguration des Testhosts und des Geräts, da diese auf allen Testhosts und Geräten nachhaltiger und konsistenter sein wird.

  Device Farm empfiehlt Ihnen zwar, Ihre Testspezifikationsdatei so weit anzupassen, wie Sie es für die Ausführung Ihrer Tests benötigen, aber die Verwaltung der Testspezifikationsdatei kann im Laufe der Zeit schwierig werden, da ihr mehr benutzerdefinierte Befehle hinzugefügt werden. Verwenden von Device Farm-verwalteter Software (über Tools wie `devicefarm-cli` und die verfügbaren Standardtools in`$PATH`) und mithilfe verwalteter Funktionen (wie dem `deviceProxy`Anforderungsparameter) zur Vereinfachung der Testspezifikationsdatei, indem die Verantwortung für die Wartung auf Device Farm selbst verlagert wird.

Testspezifikation und Testpaketcode

- Verwenden Sie keine absoluten Pfade und verlassen Sie sich nicht auf bestimmte Nebenversionen in Ihrer Testspezifikationsdatei oder Ihrem Testpaketcode. Device Farm wendet routinemäßige Updates auf den ausgewählten Testhost und die enthaltenen Softwareversionen an. Die Verwendung bestimmter oder absoluter Pfade (z. B. `/usr/local/bin/python` anstelle von`python`) oder die Anforderung bestimmter Nebenversionen (z. B. Node.js `20.3.1` statt nur `20`) kann dazu führen, dass Ihre Tests die erforderliche ausführbare Datei/nicht finden können.

  Im Rahmen der benutzerdefinierten Testausführung richtet Device Farm verschiedene Umgebungsvariablen und die `$PATH` Variable ein, um sicherzustellen, dass die Tests in unseren dynamischen Umgebungen konsistent ausgeführt werden. Weitere Informationen finden Sie unter Umgebungsvariablen für benutzerdefinierte Testumgebungen und Unterstützte Software in benutzerdefinierten Testumgebungen.
- Speichern Sie während des Testlaufs generierte oder kopierte Dateien im temporären Verzeichnis. Heute stellen wir sicher, dass der Benutzer während der Testausführung auf

das temporäre Verzeichnis (`/tmp`) zugreifen kann (neben verwalteten Verzeichnissen wie dem`$DEVICEFARM_LOG_DIR`). Andere Verzeichnisse, auf die der Benutzer Zugriff hat, können sich im Laufe der Zeit aufgrund der Anforderungen des Dienstes oder des verwendeten Betriebssystems ändern.

- Speichern Sie Ihre Testausführungsprotokolle unter`$DEVICEFARM_LOG_DIR`. Dies ist das Standardartefaktverzeichnis, das für Ihre Ausführung bereitgestellt wird, um Ausführungslogs/ Artefakte hinzuzufügen. Die von uns bereitgestellten Beispieltestspezifikationen verwenden standardmäßig dieses Verzeichnis für Artefakte.
- Stellen Sie sicher, dass Ihre Befehle bei einem Fehler während der **`test`** Phase Ihrer Testspezifikation einen Code ungleich Null zurückgeben. Wir ermitteln, ob Ihre Ausführung fehlgeschlagen ist, indem wir für jeden Shell-Befehl, der während der Phase aufgerufen wurde, nach einem Exit-Code ungleich Null suchen. `test` Sie sollten sicherstellen, dass Ihr Logik- oder Testframework für alle gewünschten Szenarien einen Exit-Code ungleich Null zurückgibt, was möglicherweise eine zusätzliche Konfiguration erfordert.

  Beispielsweise betrachten bestimmte Test-Frameworks (wie JUnit5), dass keine Tests ausgeführt wurden, nicht als Fehler, sodass festgestellt wird, dass Ihre Tests erfolgreich ausgeführt wurden, auch wenn nichts ausgeführt wurde. JUnit5 Als Beispiel müssten Sie die Befehlszeilenoption angeben, `--fail-if-no-tests` um sicherzustellen, dass dieses Szenario mit einem Exit-Code ungleich Null beendet wird.
- Überprüfen Sie die Kompatibilität der Software mit der Betriebssystemversion des Geräts und der Testhostversion, die Sie für den Testlauf verwenden werden. Beispielsweise gibt es beim Testen von Software-Frameworks (z. B. Appium) bestimmte Funktionen, die möglicherweise nicht auf allen Betriebssystemversionen des getesteten Geräts wie vorgesehen funktionieren.

Sicherheit

- Vermeiden Sie es, sensible Variablen (wie AWS-Schlüssel) in Ihrer Testspezifikationsdatei zu speichern oder zu protokollieren. Die Testspezifikationsdateien, die von der Testspezifikation generierten Skripte und die Protokolle des Testspezifikationsskripts werden am Ende der Testausführung alle als herunterladbare Artefakte bereitgestellt. Dies kann zur unbeabsichtigten Offenlegung von Geheimnissen für andere Benutzer in Ihrem Konto mit Lesezugriff auf Ihren Testlauf führen.

Migrieren von Tests von einer Standard- zu einer benutzerdefinierten Testumgebung

Sie können in AWS Device Farm von einem standardmäßigen Testausführungsmodus zu einem benutzerdefinierten Ausführungsmodus wechseln. Die Migration umfasst hauptsächlich zwei verschiedene Ausführungsformen:

1. Standardmodus: Dieser Testausführungsmodus ist in erster Linie darauf ausgelegt, Kunden detaillierte Berichte und eine vollständig verwaltete Umgebung zu bieten.
2. Benutzerdefinierter Modus: Dieser Testausführungsmodus wurde für verschiedene Anwendungsfälle entwickelt, die schnellere Testläufe, die Möglichkeit zum Lift-and-Shift-Verfahren und zur Erreichung der Parität mit der lokalen Umgebung sowie Live-Videostreaming erfordern.

Weitere Informationen zu den standardmäßigen und benutzerdefinierten Modi in Device Farm finden Sie unter Testumgebungen in AWS Device Farm undBenutzerdefinierte Testumgebungen in AWS Device Farm.

Überlegungen zur Migration

In diesem Abschnitt sind einige der wichtigsten Anwendungsfälle aufgeführt, die bei der Migration zum benutzerdefinierten Modus zu berücksichtigen sind:

1. Geschwindigkeit: Im Standardmodus analysiert Device Farm die Metadaten der Tests, die Sie gepackt und hochgeladen haben, anhand der Paketierungsanweisungen für Ihr spezielles Framework. Beim Parsen wird die Anzahl der Tests in Ihrem Paket erkannt. Danach führt Device Farm jeden Test separat aus und präsentiert die Protokolle, Videos und andere Ergebnisartefakte für jeden Test einzeln. Dies erhöht jedoch kontinuierlich die gesamte end-to-end Testausführungszeit, da die Tests und Ergebnisartefakte auf der Serviceseite vor- und nachbearbeitet werden.

   Im Gegensatz dazu analysiert der benutzerdefinierte Ausführungsmodus Ihr Testpaket nicht. Dies bedeutet, dass keine Vorverarbeitung und nur minimale Nachbearbeitung für Tests oder Ergebnisartefakte erforderlich ist. Dies führt zu einer end-to-end Gesamtausführungszeit, die Ihrer lokalen Konfiguration sehr nahe kommt. Die Tests werden in demselben Format ausgeführt, in dem sie ausgeführt würden, wenn sie auf Ihren lokalen Computern ausgeführt würden. Die Ergebnisse der Tests entsprechen denen, die Sie lokal erhalten, und stehen am Ende der Jobausführung zum Download zur Verfügung.

2. Anpassung oder Flexibilität: Der Standardausführungsmodus analysiert Ihr Testpaket, um die Anzahl der Tests zu ermitteln, und führt dann jeden Test separat aus. Beachten Sie, dass es keine Garantie dafür gibt, dass die Tests in der von Ihnen angegebenen Reihenfolge ausgeführt werden. Daher funktionieren Tests, die eine bestimmte Ausführungsreihenfolge erfordern, möglicherweise nicht wie erwartet. Darüber hinaus gibt es keine Möglichkeit, die Host-Computerumgebung anzupassen oder Konfigurationsdateien zu übergeben, die möglicherweise erforderlich sind, um Ihre Tests auf eine bestimmte Weise auszuführen.

   Im benutzerdefinierten Modus können Sie dagegen die Host-Computerumgebung konfigurieren, einschließlich der Möglichkeit, zusätzliche Software zu installieren, Filter an Ihre Tests zu übergeben, Konfigurationsdateien zu übergeben und die Einrichtung der Testausführung zu steuern. Dies wird über eine Yaml-Datei (auch als Testspec-Datei bezeichnet) erreicht, die Sie ändern können, indem Sie ihr Shell-Befehle hinzufügen. Diese Yaml-Datei wird in ein Shell-Skript konvertiert, das auf dem Test-Host-Computer ausgeführt wird. Sie können mehrere Yaml-Dateien speichern und eine dynamisch gemäß Ihren Anforderungen auswählen, wenn Sie einen Lauf planen.

3. Live-Video und Protokollierung: Sowohl der Standard- als auch der benutzerdefinierte Ausführungsmodus bieten Ihnen Videos und Protokolle für Ihre Tests. Im Standardmodus erhalten Sie das Video und die vordefinierten Protokolle Ihrer Tests jedoch erst, nachdem Ihre Tests abgeschlossen sind.

   Im benutzerdefinierten Modus erhalten Sie dagegen einen Live-Stream mit dem Video und den clientseitigen Protokollen Ihrer Tests. Darüber hinaus können Sie das Video und andere Artefakte am Ende der Tests herunterladen.

> **Tip**
>
> Wenn Ihr Anwendungsfall mindestens einen der oben genannten Faktoren beinhaltet, empfehlen wir dringend, zum benutzerdefinierten Ausführungsmodus zu wechseln.

Schritte zur Migration

Gehen Sie wie folgt vor, um vom Standardmodus zum benutzerdefinierten Modus zu migrieren:

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die Device Farm Farm-Konsole unter https://console.aws.amazon.com/devicefarm/.

2. Wählen Sie Ihr Projekt aus und starten Sie dann einen neuen Automatisierungslauf.
3. Laden Sie Ihre App hoch (oder wählen Sie sie aus`web app`), wählen Sie Ihren Testframework-Typ aus, laden Sie Ihr Testpaket hoch und wählen Sie dann unter dem `Choose your execution environment` Parameter die Option für`Run your test in a custom environment`.
4. Standardmäßig wird die Beispieldatei mit den Testspezifikationen von Device Farm angezeigt, sodass Sie sie ansehen und bearbeiten können. Diese Beispieldatei kann als Ausgangspunkt verwendet werden, um Ihre Tests im benutzerdefinierten Umgebungsmodus auszuprobieren. Sobald Sie dann von der Konsole aus überprüft haben, dass Ihre Tests ordnungsgemäß funktionieren, können Sie jede Ihrer API-, CLI- und Pipeline-Integrationen mit Device Farm ändern, um diese Testspezifikationsdatei als Parameter bei der Planung von Testläufen zu verwenden. Informationen zum Hinzufügen einer Testspezifikationsdatei als Parameter für Ihre Läufe finden Sie im `testSpecArn` Parameterabschnitt für die `ScheduleRun` API in unserem API-Leitfaden.

Appium-Framework

In einer benutzerdefinierten Testumgebung fügt Device Farm keine Appium-Funktionen in Ihre Appium-Framework-Tests ein oder überschreibt sie. Sie müssen die Appium-Funktionen Ihres Tests entweder in der YAML-Datei der Testspezifikation oder im Testcode angeben.

Android-Instrumentierung

Für die Migration Ihrer Android-Instrumentierungstests auf eine benutzerdefinierte Testumgebung müssen Sie keine Änderungen vornehmen.

iOS XCUITest

Sie müssen keine Änderungen vornehmen, um Ihre XCUITest iOS-Tests in eine benutzerdefinierte Testumgebung zu verschieben.

Erweiterung benutzerdefinierter Testumgebungen in Device Farm

AWS Device Farm ermöglicht die Konfiguration einer benutzerdefinierten Umgebung für automatisierte Tests (benutzerdefinierter Modus). Dies ist der empfohlene Ansatz für alle Device Farm Farm-Benutzer. Im benutzerdefinierten Modus von Device Farm können Sie mehr als nur Ihre Testsuite ausführen. In diesem Abschnitt erfahren Sie, wie Sie Ihre Testsuite erweitern und Ihre Tests optimieren können.

Weitere Informationen zu benutzerdefinierten Testumgebungen in Device Farm finden Sie unterBenutzerdefinierte Testumgebungen in AWS Device Farm.

Themen

- Einstellung einer Geräte-PIN beim Ausführen von Tests in Device Farm
- Beschleunigung von Appium-basierten Tests in Device Farm durch die gewünschten Funktionen
- Verwenden von Webhooks und anderen, APIs nachdem Ihre Tests in Device Farm ausgeführt wurden
- Hinzufügen zusätzlicher Dateien zu Ihrem Testpaket in Device Farm

Einstellung einer Geräte-PIN beim Ausführen von Tests in Device Farm

Bei einigen Anwendungen müssen Sie eine PIN auf dem Gerät festlegen. Device Farm unterstützt das native Einstellen einer PIN auf Geräten nicht. Dies ist jedoch mit den folgenden Einschränkungen möglich:

- Auf dem Gerät muss Android 8 oder höher ausgeführt werden.
- Die PIN muss nach Abschluss des Tests entfernt werden.

Um die PIN in Ihren Tests festzulegen, verwenden Sie die `post_test` Phasen `pre_test` und, um die PIN festzulegen und zu entfernen, wie im Folgenden gezeigt:

```
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"
```

Wenn Ihre Testsuite beginnt, ist die PIN 1234 festgelegt. Nach dem Beenden Ihrer Testsuite wird die PIN entfernt.

> **Warning**
>
> Wenn Sie die PIN nach Abschluss des Tests nicht vom Gerät entfernen, werden das Gerät und Ihr Konto unter Quarantäne gestellt.

Weitere Möglichkeiten, Ihre Testsuite zu erweitern und Ihre Tests zu optimieren, finden Sie unter. Erweiterung benutzerdefinierter Testumgebungen in Device Farm

Beschleunigung von Appium-basierten Tests in Device Farm durch die gewünschten Funktionen

Wenn Sie Appium verwenden, stellen Sie möglicherweise fest, dass die Testsuite im Standardmodus sehr langsam ist. Dies liegt daran, dass Device Farm die Standardeinstellungen anwendet und keine Annahmen darüber trifft, wie Sie die Appium-Umgebung verwenden möchten. Diese Standardeinstellungen basieren zwar auf den bewährten Methoden der Branche, gelten jedoch möglicherweise nicht für Ihre Situation. Um die Parameter des Appium-Servers zu optimieren, können Sie die Standardfunktionen von Appium in Ihrer Testspezifikation anpassen. Im Folgenden wird beispielsweise die `usePrebuildWDA` Fähigkeit `true` in einer iOS-Testsuite auf festgelegt, um die anfängliche Startzeit zu beschleunigen:

```
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 &
```

Bei Appium-Fähigkeiten muss es sich um eine JSON-Struktur mit Shell-Escape-Code in Anführungszeichen handeln.

Die folgenden Appium-Funktionen sind häufig Quellen für Leistungsverbesserungen:

`noReset` und `fullReset`

Diese beiden Funktionen, die sich gegenseitig ausschließen, beschreiben das Verhalten von Appium nach Abschluss jeder Sitzung. Wenn auf gesetzt `noReset` ist`true`, entfernt der Appium-Server keine Daten aus Ihrer Anwendung, wenn eine Appium-Sitzung endet, und führt praktisch keine Bereinigung durch. `fullReset`deinstalliert und löscht alle Anwendungsdaten vom Gerät, nachdem die Sitzung geschlossen wurde. Weitere Informationen finden Sie unter Reset-Strategien in der Appium-Dokumentation.

`ignoreUnimportantViews`(Nur Android)

Weist Appium an, die Hierarchie der Android-Benutzeroberfläche nur auf die für den Test relevanten Ansichten zu komprimieren, wodurch die Suche nach bestimmten Elementen beschleunigt wird. Dies kann jedoch einige XPath basierte Testsuiten beschädigen, da die Hierarchie des UI-Layouts geändert wurde.

`skipUnlock`(Nur Android)

Informiert Appium darüber, dass derzeit kein PIN-Code festgelegt ist, was Tests nach einem Ausschalten des Bildschirms oder einem anderen Sperrenereignis beschleunigt.

`webDriverAgentUrl`(nur iOS)

Weist Appium an, davon auszugehen, dass eine wichtige iOS-Abhängigkeit,`webDriverAgent`, bereits läuft und für die Annahme von HTTP-Anfragen an der angegebenen URL verfügbar ist. Wenn Appium noch `webDriverAgent` nicht aktiv ist, kann es zu Beginn einer Testsuite einige Zeit dauern, bis Appium gestartet ist. `webDriverAgent` Wenn Sie `webDriverAgent` selbst starten und `http://localhost:8100` beim Start von Appium `webDriverAgentUrl` auf einstellen, können Sie Ihre Testsuite schneller starten. Beachten Sie, dass diese Funktion niemals zusammen mit der `useNewWDA` Funktion verwendet werden sollte.

Sie können den folgenden Code verwenden, um mit Ihrer Testspezifikationsdatei am lokalen Port `8100` des Geräts zu beginnen `webDriverAgent` und sie dann an den lokalen Port des Testhosts weiterzuleiten `8100` (auf diese Weise können Sie den Wert auf setzen `webDriverAgentUrlhttp://localhost:8100`). Dieser Code sollte während der Installationsphase ausgeführt werden, nachdem der Code für die Einrichtung des Appium und der `webDriverAgent` Umgebungsvariablen definiert wurde:

```
      # 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 &
```

Anschließend können Sie Ihrer Testspezifikationsdatei den folgenden Code hinzufügen, um sicherzustellen, dass sie erfolgreich `webDriverAgent` gestartet wurde. Dieser Code sollte am Ende der Vortestphase ausgeführt werden, nachdem sichergestellt wurde, dass Appium erfolgreich gestartet wurde:

```
      # 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;
```

Weitere Informationen zu den Funktionen, die Appium unterstützt, finden Sie unter Appium Desired Capabilities in der Appium-Dokumentation.

Weitere Möglichkeiten, Ihre Testsuite zu erweitern und Ihre Tests zu optimieren, finden Sie unter. Erweiterung benutzerdefinierter Testumgebungen in Device Farm

Verwenden von Webhooks und anderen, APIs nachdem Ihre Tests in Device Farm ausgeführt wurden

Sie können Device Farm einen Webhook aufrufen lassen, nachdem die Verwendung curl jeder Testsuite abgeschlossen ist. Der dazu erforderliche Vorgang hängt vom Ziel und der Formatierung ab. Informationen zu Ihrem spezifischen Webhook finden Sie in der Dokumentation zu diesem Webhook. Im folgenden Beispiel wird jedes Mal, wenn eine Testsuite fertig ist, eine Nachricht an einen Slack-Webhook gesendet:

```
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
```

Weitere Informationen zur Verwendung von Webhooks mit Slack findest du unter Deine erste Slack-Nachricht mit Webhook versenden in der Slack-API-Referenz.

Weitere Möglichkeiten, deine Testsuite zu erweitern und deine Tests zu optimieren, findest du unter. Erweiterung benutzerdefinierter Testumgebungen in Device Farm

Sie sind nicht darauf beschränkt, Webhooks curl aufzurufen. Testpakete können zusätzliche Skripts und Tools enthalten, sofern sie mit der Device Farm Farm-Ausführungsumgebung kompatibel sind. Ihr Testpaket kann beispielsweise Hilfsskripten enthalten, die Anfragen an andere richten APIs. Stellen Sie sicher, dass alle erforderlichen Pakete zusammen mit den Anforderungen Ihrer Testsuite installiert sind. Um ein Skript hinzuzufügen, das ausgeführt wird, nachdem Ihre Testsuite fertiggestellt ist, nehmen Sie das Skript in Ihr Testpaket auf und fügen Sie Ihrer Testspezifikation Folgendes hinzu:

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

> **Note**
>
> Die Verwaltung aller API-Schlüssel oder anderer Authentifizierungstoken, die in Ihrem Testpaket verwendet werden, liegt in Ihrer Verantwortung. Wir empfehlen Ihnen, jegliche Form von Sicherheitsanmeldedaten außerhalb der Quellcodeverwaltung aufzubewahren, Anmeldeinformationen mit möglichst wenigen Rechten zu verwenden und wann immer möglich widerrufbare, kurzlebige Token zu verwenden. Informationen zur Überprüfung der Sicherheitsanforderungen finden Sie in der Dokumentation des Drittanbieters, den Sie verwenden. APIs

Wenn Sie beabsichtigen, AWS Dienste als Teil Ihrer Testausführungssuite zu verwenden, sollten Sie temporäre IAM-Anmeldeinformationen verwenden, die außerhalb Ihrer Testsuite generiert wurden und in Ihrem Testpaket enthalten sind. Für diese Anmeldeinformationen sollten so wenige Berechtigungen wie möglich erteilt werden und die Lebensdauer sollte so kurz wie möglich sein. Weitere Informationen zum Erstellen temporärer Anmeldeinformationen finden Sie unter Temporäre Sicherheitsanmeldeinformationen anfordern im IAM-Benutzerhandbuch.

Weitere Möglichkeiten, Ihre Testsuite zu erweitern und Ihre Tests zu optimieren, finden Sie unterErweiterung benutzerdefinierter Testumgebungen in Device Farm.

Hinzufügen zusätzlicher Dateien zu Ihrem Testpaket in Device Farm

Möglicherweise möchten Sie zusätzliche Dateien als Teil Ihrer Tests verwenden, entweder als zusätzliche Konfigurationsdateien oder als zusätzliche Testdaten. Sie können diese zusätzlichen Dateien Ihrem Testpaket hinzufügen, bevor Sie es hochladen AWS Device Farm, und dann im benutzerdefinierten Umgebungsmodus darauf zugreifen. Grundsätzlich handelt es sich bei allen Uploadformaten für Testpakete (ZIP, IPA, APK, JAR usw.) um Paketarchivformate, die standardmäßige ZIP-Operationen unterstützen.

Sie können Ihrem Testarchiv Dateien hinzufügen, bevor Sie es hochladen, AWS Device Farm indem Sie den folgenden Befehl verwenden:

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

Für ein Verzeichnis mit zusätzlichen Dateien:

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

Diese Befehle funktionieren erwartungsgemäß für alle Upload-Formate von Testpaketen mit Ausnahme von IPA-Dateien. Für IPA-Dateien, insbesondere wenn sie mit verwendet werden, empfehlen wir XCUITests, dass Sie alle zusätzlichen Dateien aufgrund der Art und Weise, wie AWS Device Farm iOS-Testpakete entworfen werden, an einem etwas anderen Speicherort ablegen. Beim Erstellen Ihres iOS-Tests befindet sich das Testanwendungsverzeichnis in einem anderen Verzeichnis mit dem Namen*Payload*.

So könnte beispielsweise ein solches iOS-Testverzeichnis aussehen:

```
$ 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
```

Fügen Sie für diese XCUITest Pakete alle zusätzlichen Dateien zu dem Verzeichnis hinzu, das *.app* innerhalb des *Payload* Verzeichnisses endet. Die folgenden Befehle zeigen beispielsweise, wie Sie diesem Testpaket eine Datei hinzufügen können:

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

Wenn Sie Ihrem Testpaket eine Datei hinzufügen, können Sie AWS Device Farm je nach Upload-Format ein leicht unterschiedliches Interaktionsverhalten erwarten. Wenn für den Upload die ZIP-Dateierweiterung verwendet wurde, AWS Device Farm wird der Upload vor dem Test automatisch entpackt und die entpackten Dateien werden an dem Speicherort mit der *$DEVICEFARM_TEST_PACKAGE_PATH* Umgebungsvariablen belassen. (Das heißt, wenn Sie wie im ersten Beispiel eine Datei mit dem Namen *extra_file* zum Stammverzeichnis des Archivs hinzufügen würden, würde sie sich *$DEVICEFARM_TEST_PACKAGE_PATH/extra_file* während des Tests unter befinden).

Um ein praktischeres Beispiel zu verwenden: Wenn Sie ein Appium TestNG-Benutzer sind und eine *testng.xml* Datei in Ihren Test aufnehmen möchten, können Sie sie mit dem folgenden Befehl in Ihr Archiv aufnehmen:

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

Anschließend können Sie Ihren Testbefehl im benutzerdefinierten Umgebungsmodus wie folgt ändern:

```
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
```

Wenn Ihre Upload-Erweiterung für das Testpaket nicht ZIP ist (z. B. APK-, IPA- oder JAR-Datei), finden Sie die hochgeladene Paketdatei selbst unter*$DEVICEFARM_TEST_PACKAGE_PATH*. Da es sich immer noch um Dateien im Archivformat handelt, können Sie die Datei entpacken, um von innen auf die zusätzlichen Dateien zuzugreifen. Mit dem folgenden Befehl wird beispielsweise der Inhalt des Testpakets (für APK-, IPA- oder JAR-Dateien) in das Verzeichnis entpackt: */tmp*

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

Im Fall einer APK- oder JAR-Datei würden Sie feststellen, dass Ihre zusätzlichen Dateien in das *`/tmp`* Verzeichnis entpackt wurden (z. B.). *`/tmp/extra_file`* Im Fall einer IPA-Datei würden sich zusätzliche Dateien, wie bereits erläutert, an einem etwas anderen Ort innerhalb des Ordners befinden, der auf*`.app`*, endet und sich innerhalb des Verzeichnisses befindet. *`Payload`* Basierend auf dem obigen IPA-Beispiel würde sich die Datei beispielsweise an dem Speicherort befinden */`tmp/Payload/ADFiOSReferenceAppUITests-Runner.app/extra_file`* (referenzierbar als). */`tmp/Payload/*.app/extra_file`*

Weitere Möglichkeiten, Ihre Testsuite zu erweitern und Ihre Tests zu optimieren, finden Sie unter. Erweiterung benutzerdefinierter Testumgebungen in Device Farm

Fernzugriff in AWS Device Farm

Der Remotezugriff ermöglicht, Geräte über Ihren Webbrowser in Echtzeit zu bedienen, inklusive Fingergesten, Gesten und anderen Interaktionen, beispielsweise, um Funktionalitäten zu testen und Kundenprobleme zu reproduzieren. Sie interagieren mit einem bestimmten Gerät, indem Sie ein Remotezugriffssitzung mit diesem Gerät erstellen.

Eine Sitzung in Device Farm ist eine Echtzeitinteraktion mit einem tatsächlichen, physischen Gerät, das in einem Webbrowser gehostet wird. In einer Sitzung wird das einzelne Gerät angezeigt, das Sie beim Start der Sitzung ausgewählt haben. Ein Benutzer kann mehrere Sitzungen gleichzeitig starten, wobei die Gesamtanzahl von gleichzeitigen Geräten durch die Anzahl der Geräteplätze, die Sie zur Verfügung haben, beschränkt wird. Sie können Geräteplätze für bestimmte Gerätefamilien (Android- oder iOS-Geräte) erwerben. Weitere Informationen finden Sie unter Device Farm – Preise.

Device Farm bietet derzeit eine Untergruppe von Geräten für Fernzugriffstests an. Diesem Gerätepool werden laufend neue Geräte hinzugefügt.

Device Farm zeichnet Videos von jeder Fernzugriffssitzung auf und generiert während der Sitzung Aktivitätsprotokolle. Diese Ergebnisse enthalten alle Informationen, die Sie während einer Sitzung eingeben.

> **Note**
>
> Aus Sicherheitsgründen empfehlen wir Ihnen, in einer Remotezugriffssitzung die Angabe oder Eingabe von sensiblen Daten, wie beispielsweise Kontonummern, persönlichen Anmeldeinformationen und anderen Details, zu vermeiden. Verwenden Sie nach Möglichkeit Alternativen, die speziell für Tests entwickelt wurden, z. B. Testkonten.

Themen

Erstellen einer Fernzugriffssitzung in AWS Device Farm

Weitere Informationen zu Remotezugriffssitzungen finden Sie unter Sitzungen.

Voraussetzungen

- Erstellen Sie ein Projekt in Device Farm. Befolgen Sie die Anweisungen unter Ein Projekt in AWS Device Farm erstellen, und kehren Sie dann zu dieser Seite zurück.

Erstellen Sie eine Remotesitzung

Console

1. Melden Sie sich bei der Device Farm Farm-Konsole unter https://console.aws.amazon.com/devicefarm an.
2. Wählen Sie im Navigationsbereich Device Farm die Option Mobile Device Testing und dann Projects aus.
3. Wenn Sie bereits ein Projekt haben, wählen Sie es aus der Liste aus. Andernfalls erstellen Sie ein Projekt, indem Sie den Anweisungen unter folgenEin Projekt in AWS Device Farm erstellen.
4. Wählen Sie auf der Registerkarte Fernzugriff die Option Fernzugriffssitzung erstellen aus.
5. Wählen Sie ein Gerät für Ihre Sitzung. Sie können aus der Liste der verfügbaren Geräte auswählen oder mithilfe der Suchleiste oben in der Liste nach einem Gerät suchen.
6. (Optional) Nehmen Sie eine App und zusätzliche Apps in die Sitzung auf. Dabei kann es sich um neu hochgeladene Apps oder um Apps handeln, die zuvor in den letzten 30 Tagen in dieses Projekt hochgeladen wurden (nach 30 Tagen laufen App-Uploads ab).
7. Geben Sie unter Session name (Sitzungsname) einen Namen für die Sitzung ein.
8. Wählen Sie Confirm and start session (Bestätigen und Sitzung starten).

AWS CLI

Hinweis: Diese Anweisungen beziehen sich nur auf die Erstellung einer Fernzugriffssitzung. Anweisungen zum Hochladen einer App zur Verwendung während Ihrer Sitzung finden Sie unter Automatisieren von App-Uploads.

Stellen Sie zunächst sicher, dass Ihre AWS-CLI-Version vorhanden ist, up-to-date indem Sie die neueste Version herunterladen und installieren.

> **Important**
>
> Bestimmte in diesem Dokument erwähnte Befehle sind in älteren Versionen der AWS-CLI nicht verfügbar.

Anschließend können Sie bestimmen, auf welchem Gerät Sie testen möchten:

```
$ aws devicefarm list-devices
```

Dadurch werden Ausgaben wie die folgende angezeigt:

```
{
    "devices":
    [
        {
            "arn": "arn:aws:devicefarm:us-
west-2::device:DE5BD47FF3BD42C3A14BF7A6EFB1BFE7",
            "name": "Google Pixel 8",
            "remoteAccessEnabled": true,
            "availability": "HIGHLY_AVAILABLE"
            ...
        },
        ...
    ]
}
```

Anschließend können Sie Ihre Fernzugriffssitzung mit einem Geräte-ARN Ihrer Wahl erstellen:

```
$ aws devicefarm create-remote-access-session \
  --project-arn arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef \
  --device-arn arn:aws:devicefarm:us-west-2::device:DE5BD47FF3BD42C3A14BF7A6EFB1BFE7
 \
  --app-arn arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
 \
  --configuration '{
      "auxiliaryApps": [
```

```
          "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
          "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
      ]
  }'
```

Dadurch werden Ausgaben wie die folgende angezeigt:

```
{
    "remoteAccessSession": {
        "arn": "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000",
        "name": "Google Pixel 8",
        "status": "PENDING",
        ...
}
```

Jetzt können wir optional eine Umfrage durchführen und warten, bis die Sitzung fertig ist:

```
$ POLL_INTERVAL=3
TIMEOUT=600
DEADLINE=$(( $(date +%s) + TIMEOUT ))

while [[ "$(date +%s)" -lt "$DEADLINE" ]]; do

  STATUS=$(aws devicefarm get-remote-access-session \
    --arn "$DEVICE_FARM_SESSION_ARN" \
    --query 'remoteAccessSession.status' \
    --output text)

  case "$STATUS" in
    RUNNING)
      echo "Session is ready with status: $STATUS"
      break
      ;;
    STOPPING|COMPLETED)
      echo "Session ended early with status: $STATUS"
      exit 1
      ;;
  esac
```

```
done
```

Python

Hinweis: Diese Anweisungen konzentrieren sich nur auf das Erstellen einer Fernzugriffssitzung. Anweisungen zum Hochladen einer App zur Verwendung während Ihrer Sitzung finden Sie unter Automatisieren von App-Uploads.

In diesem Beispiel wird zuerst nach allen verfügbaren Google Pixel-Geräten auf Device Farm gesucht, dann eine Fernzugriffssitzung mit diesem Gerät erstellt und gewartet, bis die Sitzung ausgeführt wird.

```
import random
import time
import boto3

client = boto3.client("devicefarm", region_name="us-west-2")

# 1) Gather all matching devices via paginated ListDevices with filters
filters = [
    {"attribute": "MODEL",        "operator": "CONTAINS", "values": ["Pixel"]},
    {"attribute": "AVAILABILITY", "operator": "EQUALS",   "values": ["AVAILABLE"]},
]

matching_arns = []
next_token = None
while True:
    args = {"filters": filters}
    if next_token:
        args["nextToken"] = next_token
    page = client.list_devices(**args)
    for d in page.get("devices", []):
        matching_arns.append(d["arn"])
    next_token = page.get("nextToken")
    if not next_token:
        break

if not matching_arns:
    raise RuntimeError("No available Google Pixel device found.")

# Randomly select one device from the full matching set
device_arn = random.choice(matching_arns)
print("Selected device ARN:", device_arn)
```

```
# 2) Create remote access session and wait until RUNNING
resp = client.create_remote_access_session(
    projectArn="arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef",
    deviceArn=device_arn,
    appArn="arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
  # optional
    configuration={
        "auxiliaryApps": [  # optional
            "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
            "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
        ]
    },
)

session_arn = resp["remoteAccessSession"]["arn"]
print(f"Created Remote Access Session: {session_arn}")

poll_interval = 3
timeout = 600
deadline = time.time() + timeout
terminal_states = ["STOPPING", "COMPLETED"]

while True:
    out = client.get_remote_access_session(arn=session_arn)
    status = out["remoteAccessSession"]["status"]
    print(f"Current status: {status}")

    if status == "RUNNING":
        print(f"Session is ready with status: {status}")
        break
    if status in terminal_states:
        raise RuntimeError(f"Session ended early with status: {status}")
    if time.time() >= deadline:
        raise RuntimeError("Timed out waiting for session to be ready.")
    time.sleep(poll_interval)
```

Java

Hinweis: Diese Anweisungen beziehen sich nur auf das Erstellen einer Fernzugriffssitzung. Anweisungen zum Hochladen einer App zur Verwendung während Ihrer Sitzung finden Sie unter Automatisieren von App-Uploads.

Hinweis: Dieses Beispiel verwendet das AWS SDK for Java v2 und ist mit JDK-Versionen 11 und höher kompatibel.

In diesem Beispiel wird zuerst nach allen verfügbaren Google Pixel-Geräten auf Device Farm gesucht, dann eine Fernzugriffssitzung mit diesem Gerät erstellt und gewartet, bis die Sitzung ausgeführt wird.

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.concurrent.ThreadLocalRandom;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import
 software.amazon.awssdk.services.devicefarm.model.CreateRemoteAccessSessionConfiguration;
import
 software.amazon.awssdk.services.devicefarm.model.CreateRemoteAccessSessionRequest;
import
 software.amazon.awssdk.services.devicefarm.model.CreateRemoteAccessSessionResponse;
import software.amazon.awssdk.services.devicefarm.model.Device;
import software.amazon.awssdk.services.devicefarm.model.DeviceFilter;
import software.amazon.awssdk.services.devicefarm.model.DeviceFilterAttribute;
import
 software.amazon.awssdk.services.devicefarm.model.GetRemoteAccessSessionRequest;
import
 software.amazon.awssdk.services.devicefarm.model.GetRemoteAccessSessionResponse;
import software.amazon.awssdk.services.devicefarm.model.ListDevicesRequest;
import software.amazon.awssdk.services.devicefarm.model.ListDevicesResponse;
import software.amazon.awssdk.services.devicefarm.model.RuleOperator;

public class CreateRemoteAccessSession {
  public static void main(String[] args) throws Exception {
    DeviceFarmClient client = DeviceFarmClient.builder()
        .region(Region.US_WEST_2)
        .build();
```

```
    String projectArn = "arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef";
    String appArn     = "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
    String aux1       = "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
    String aux2       = "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789

    // 1) Gather all matching devices via paginated ListDevices with filters
    List<DeviceFilter> filters = Arrays.asList(
        DeviceFilter.builder()
            .attribute(DeviceFilterAttribute.MODEL)
            .operator(RuleOperator.CONTAINS)
            .values("Pixel")
            .build(),
        DeviceFilter.builder()
            .attribute(DeviceFilterAttribute.AVAILABILITY)
            .operator(RuleOperator.EQUALS)
            .values("AVAILABLE")
            .build()
    );

    List<String> matchingDeviceArns = new ArrayList<>();
    String next = null;
    do {
      ListDevicesResponse page = client.listDevices(
          ListDevicesRequest.builder().filters(filters).nextToken(next).build());
      for (Device d : page.devices()) {
        matchingDeviceArns.add(d.arn());
      }
      next = page.nextToken();
    } while (next != null);

    if (matchingDeviceArns.isEmpty()) {
      throw new RuntimeException("No available Google Pixel device found.");
    }

    // Randomly select one device from the full matching set
    String deviceArn = matchingDeviceArns.get(
        ThreadLocalRandom.current().nextInt(matchingDeviceArns.size()));
    System.out.println("Selected device ARN: " + deviceArn);

    // 2) Create Remote Access session and wait until it is RUNNING
```

```
    CreateRemoteAccessSessionConfiguration cfg =
 CreateRemoteAccessSessionConfiguration.builder()
        .auxiliaryApps(Arrays.asList(aux1, aux2))
        .build();

    CreateRemoteAccessSessionResponse res = client.createRemoteAccessSession(
        CreateRemoteAccessSessionRequest.builder()
            .projectArn(projectArn)
            .deviceArn(deviceArn)
            .appArn(appArn)       // optional
            .configuration(cfg)   // optional
            .build());

    String sessionArn = res.remoteAccessSession().arn();
    System.out.println("Created Remote Access Session: " + sessionArn);

    int pollIntervalMs = 3000;
    long timeoutMs = 600_000L;
    long deadline = System.currentTimeMillis() + timeoutMs;

    while (true) {
      GetRemoteAccessSessionResponse get = client.getRemoteAccessSession(
          GetRemoteAccessSessionRequest.builder().arn(sessionArn).build());
      String status = get.remoteAccessSession().statusAsString();
      System.out.println("Current status: " + status);

      if ("RUNNING".equals(status)) {
        System.out.println("Session is ready with status: " + status);
        break;
      }
      if ("STOPPING".equals(status) || "COMPLETED".equals(status)) {
        throw new RuntimeException("Session ended early with status: " + status);
      }
      if (System.currentTimeMillis() >= deadline) {
        throw new RuntimeException("Timed out waiting for session to be ready.");
      }
      Thread.sleep(pollIntervalMs);
    }
  }
}
```

JavaScript

Hinweis: Diese Anweisungen beziehen sich nur auf das Erstellen einer Fernzugriffssitzung. Anweisungen zum Hochladen einer App zur Verwendung während Ihrer Sitzung finden Sie unter Automatisieren von App-Uploads.

Hinweis: In diesem Beispiel wird das AWS-SDK für JavaScript Version 3 verwendet.

In diesem Beispiel wird zuerst nach allen verfügbaren Google Pixel-Geräten auf Device Farm gesucht, dann eine Fernzugriffssitzung mit diesem Gerät erstellt und gewartet, bis die Sitzung ausgeführt wird.

```
import {
  DeviceFarmClient,
  ListDevicesCommand,
  CreateRemoteAccessSessionCommand,
  GetRemoteAccessSessionCommand,
} from "@aws-sdk/client-device-farm";

const client = new DeviceFarmClient({ region: "us-west-2" });

// 1) Gather all matching devices via paginated ListDevices with filters
const filters = [
  { attribute: "MODEL",        operator: "CONTAINS", values: ["Pixel"] },
  { attribute: "AVAILABILITY", operator: "EQUALS",   values: ["AVAILABLE"] },
];

let nextToken;
const matching = [];

while (true) {
  const page = await client.send(new ListDevicesCommand({ filters, nextToken }));
  for (const d of page.devices ?? []) {
    matching.push(d.arn);
  }
  nextToken = page.nextToken;
  if (!nextToken) break;
}

if (matching.length === 0) {
  throw new Error("No available Google Pixel device found.");
}
```

```
// Randomly select one device from the full matching set
const deviceArn = matching[Math.floor(Math.random() * matching.length)];
console.log("Selected device ARN:", deviceArn);

// 2) Create remote access session and wait until RUNNING
const out = await client.send(new CreateRemoteAccessSessionCommand({
  projectArn: "arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef",
  deviceArn,
  appArn: "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
 optional
  configuration: {
    auxiliaryApps: [ // optional
      "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
      "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
    ],
  },
}));

const sessionArn = out.remoteAccessSession?.arn;
console.log("Created Remote Access Session:", sessionArn);

const pollIntervalMs = 3000;
const timeoutMs = 600000;
const deadline = Date.now() + timeoutMs;

while (true) {
  const get = await client.send(new GetRemoteAccessSessionCommand({ arn:
 sessionArn }));
  const status = get.remoteAccessSession?.status;
  console.log("Current status:", status);

  if (status === "RUNNING") {
    console.log("Session is ready with status:", status);
    break;
  }
  if (status === "STOPPING" || status === "COMPLETED") {
    throw new Error(`Session ended early with status: ${status}`);
  }
  if (Date.now() >= deadline) {
    throw new Error("Timed out waiting for session to be ready.");
```

```
  }
  await new Promise((r) => setTimeout(r, pollIntervalMs));
}
```

C#

Hinweis: Diese Anweisungen beziehen sich nur auf das Erstellen einer Fernzugriffssitzung. Anweisungen zum Hochladen einer App zur Verwendung während Ihrer Sitzung finden Sie unter Automatisieren von App-Uploads.

In diesem Beispiel wird zuerst nach allen verfügbaren Google Pixel-Geräten auf Device Farm gesucht, dann eine Fernzugriffssitzung mit diesem Gerät erstellt und gewartet, bis die Sitzung ausgeführt wird.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

class Program
{
    static async Task Main()
    {
        var client = new AmazonDeviceFarmClient(RegionEndpoint.USWest2);

        // 1) Gather all matching devices via paginated ListDevices with filters
        var filters = new List<DeviceFilter>
        {
            new DeviceFilter { Attribute = DeviceAttribute.MODEL,        Operator =
 RuleOperator.CONTAINS, Values = new List<string>{ "Pixel" } },
            new DeviceFilter { Attribute = DeviceAttribute.AVAILABILITY, Operator =
 RuleOperator.EQUALS,   Values = new List<string>{ "AVAILABLE" } },
        };

        var matchingArns = new List<string>();
        string nextToken = null;

        do
        {
            var list = await client.ListDevicesAsync(new ListDevicesRequest
            {
```

```
                Filters = filters,
                NextToken = nextToken
            });

            foreach (var d in list.Devices)
                matchingArns.Add(d.Arn);

            nextToken = list.NextToken;
        }
        while (nextToken != null);

        if (matchingArns.Count == 0)
            throw new Exception("No available Google Pixel device found.");

        // Randomly select one device from the full matching set
        var rnd = new Random();
        var deviceArn = matchingArns[rnd.Next(matchingArns.Count)];
        Console.WriteLine($"Selected device ARN: {deviceArn}");

        // 2) Create remote access session and wait until RUNNING
        var request = new CreateRemoteAccessSessionRequest
        {
            ProjectArn = "arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef",
            DeviceArn  = deviceArn,
            AppArn     = "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
 optional
            Configuration = new CreateRemoteAccessSessionConfiguration
            {
                AuxiliaryApps = new List<string>
                {
                    "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
                    "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
                }
            }
        };

        request.Configuration.AuxiliaryApps.RemoveAll(string.IsNullOrWhiteSpace);

        var response = await client.CreateRemoteAccessSessionAsync(request);
        var sessionArn = response.RemoteAccessSession.Arn;
```

```
        Console.WriteLine($"Created Remote Access Session: {sessionArn}");

        var pollIntervalMs = 3000;
        var timeoutMs = 600000;
        var deadline = DateTime.UtcNow.AddMilliseconds(timeoutMs);

        while (true)
        {
            var get = await client.GetRemoteAccessSessionAsync(new
 GetRemoteAccessSessionRequest { Arn = sessionArn });
            var status = get.RemoteAccessSession.Status.Value;
            Console.WriteLine($"Current status: {status}");

            if (status == "RUNNING")
            {
                Console.WriteLine($"Session is ready with status: {status}");
                break;
            }
            if (status == "STOPPING" || status == "COMPLETED")
            {
                throw new Exception($"Session ended early with status: {status}");
            }
            if (DateTime.UtcNow >= deadline)
            {
                throw new TimeoutException("Timed out waiting for session to be
 ready.");
            }

            await Task.Delay(pollIntervalMs);
        }
    }
}
```

Ruby

Hinweis: Diese Anweisungen beziehen sich nur auf das Erstellen einer Fernzugriffssitzung. Anweisungen zum Hochladen einer App zur Verwendung während Ihrer Sitzung finden Sie unter Automatisieren von App-Uploads.

In diesem Beispiel wird zuerst nach allen verfügbaren Google Pixel-Geräten auf Device Farm gesucht, dann eine Fernzugriffssitzung mit diesem Gerät erstellt und gewartet, bis die Sitzung ausgeführt wird.

```
require 'aws-sdk-devicefarm'

client = Aws::DeviceFarm::Client.new(region: 'us-west-2')

# 1) Gather all matching devices via paginated ListDevices with filters
filters = [
  { attribute: 'MODEL',        operator: 'CONTAINS', values: ['Pixel'] },
  { attribute: 'AVAILABILITY', operator: 'EQUALS',   values: ['AVAILABLE'] },
]

matching_arns = []
next_token = nil
loop do
  resp = client.list_devices(filters: filters, next_token: next_token)
  resp.devices&.each { |d| matching_arns << d.arn }
  next_token = resp.next_token
  break unless next_token
end

abort "No available Google Pixel device found." if matching_arns.empty?

# Randomly select one device from the full matching set
device_arn = matching_arns.sample
puts "Selected device ARN: #{device_arn}"

# 2) Create remote access session and wait until RUNNING
resp = client.create_remote_access_session(
  project_arn: "arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef",
  device_arn:  device_arn,
  app_arn:     "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
 # optional
  configuration: {
    auxiliary_apps: [ # optional
      "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
      "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
    ].compact
  }
)
```

```
session_arn = resp.remote_access_session.arn
puts "Created Remote Access Session: #{session_arn}"

poll_interval = 3
timeout = 600
deadline = Time.now + timeout
terminal = %w[STOPPING COMPLETED]

loop do
  get = client.get_remote_access_session(arn: session_arn)
  status = get.remote_access_session.status
  puts "Current status: #{status}"

  if status == 'RUNNING'
    puts "Session is ready with status: #{status}"
    break
  end

  abort "Session ended early with status: #{status}" if terminal.include?(status)
  abort "Timed out waiting for session to be ready." if Time.now >= deadline
  sleep poll_interval
end
```

Nächste Schritte

Device Farm startet die Sitzung, sobald das angeforderte Gerät und die angeforderte Infrastruktur verfügbar sind, normalerweise innerhalb weniger Minuten. Das Dialogfeld Gerät angefordert wird angezeigt, bis die Sitzung gestartet wird. Um die Sitzungsanfrage abzubrechen, wählen Sie Cancel request (Anforderung abbrechen).

Wenn das ausgewählte Gerät nicht verfügbar oder ausgelastet ist, wird der Sitzungsstatus als Ausstehendes Gerät angezeigt, was darauf hinweist, dass Sie möglicherweise einige Zeit warten müssen, bis das Gerät zum Testen verfügbar ist.

Wenn Ihr Konto das Parallelitätslimit für Geräte mit oder ohne Messgerät erreicht hat, wird der Sitzungsstatus als Gleichzeitigkeit ausstehend angezeigt. Bei Geräteslots ohne Zähler können Sie die Parallelität erhöhen, indem Sie weitere Geräteslots kaufen. Für pay-as-you-go Geräte mit Messgerät wenden Sie sich bitte über ein Support-Ticket an AWS, um eine Erhöhung des Servicekontingents zu beantragen.

Wenn das Sitzungs-Setup beginnt, wird zunächst der Status In Bearbeitung und dann der Status Verbindung hergestellt angezeigt, während Ihr lokaler Webbrowser versucht, eine Fernverbindung zum Gerät herzustellen.

Wenn Sie nach dem Start einer Sitzung den Browser oder die Browser-Registerkarte schließen, ohne die Sitzung anzuhalten, oder wenn die Verbindung zwischen dem Browser und dem Internet verloren geht, bleibt die Sitzung für fünf weitere Minuten aktiv. Danach beendet Device Farm die Sitzung. Ihr Konto wird jedoch für die Leerlaufzeit belastet.

Nach dem Start der Sitzung können Sie im Webbrowser mit dem Gerät interagieren oder das Gerät mit Appium testen.

Verwenden einer Fernzugriffssitzung in AWS Device Farm

Weitere Informationen zur Ausführung von interaktiven Tests mit Android- und iOS-Apps über Remotezugriffssitzungen finden Sie unter Sitzungen.

Voraussetzungen

- Erstellen Sie eine Sitzung. Befolgen Sie die Anweisungen unter Erstellen einer Sitzung, und kehren Sie dann zu dieser Seite zurück.

Verwenden Sie eine Sitzung in der Device Farm Farm-Konsole

Sobald das von Ihnen für eine Remotezugriffssitzung angeforderte Gerät verfügbar ist, zeigt die Konsole den Gerätebildschirm an. Die Sitzung dauert maximal 150 Minuten. Die verbleibende Zeit der Sitzung wird im Feld Restzeit neben dem Gerätenamen angezeigt.

Eine Anwendung installieren

Um eine Anwendung auf dem Sitzungsgerät zu installieren, wählen Sie unter Anwendungen installieren die Option Datei auswählen und wählen Sie dann die APK-Datei (Android) oder die IPA-

Datei (iOS) aus, die Sie installieren möchten. Anwendungen, die Sie in einer Remotezugriffssitzung ausführen, erfordern keine Testinstrumentierung oder -bereitstellung.

> **Note**
>
> Wenn Sie eine App hochladen, gibt es manchmal eine Verzögerung, bevor die App verfügbar ist. Es erscheint eine Bestätigungsmeldung, die Sie darüber informiert, ob die App erfolgreich installiert wurde oder nicht.

Das Gerät steuern

Sie können mit dem in der Konsole angezeigten Gerät wie mit dem echten physischen Gerät interagieren, indem Sie Ihre Maus oder ein ähnliches Gerät für die Touch-Eingabe und die Bildschirmtastatur des Geräts verwenden. Im Fall von Android-Geräten gibt es Schaltflächen in View controls (Steuerelemente anzeigen), die genau wie die Schaltflächen Start und Zurück auf einem Android-Gerät funktionieren. Im Fall von iOS-Geräten gibt es eine Schaltfläche namens Home, die genauso wie die Startschaltfläche auf einem iOS-Gerät funktioniert. Sie können auch zwischen Anwendungen wechseln, die auf dem Gerät ausgeführt werden, indem Sie Letzte Apps wählen.

Zwischen Hoch- und Querformat wechseln

Sie können für die Geräte, die Sie verwenden, auch zwischen dem Hochformat- (vertikal) und dem Querformat (horizontal) wechseln.

Nächste Schritte

Device Farm setzt die Sitzung fort, bis Sie sie manuell beenden oder das Zeitlimit von 150 Minuten erreicht ist. Um die Sitzung zu beenden, wählen Sie Sitzung beenden. Nachdem die Sitzung beendet wurde, können Sie auf das erfasste Video und die erstellten Protokolle zugreifen. Weitere Informationen finden Sie unter Sitzungsergebnisse werden abgerufen.

Tipps und Tricks

In einigen AWS Regionen können Leistungsprobleme bei der Fernzugriffssitzung auftreten. Dies liegt teilweise an der Latenz in einigen Regionen. Wenn Leistungsprobleme auftreten, lassen Sie der Remotesitzung etwas Zeit, um aufzuholen, bevor Sie erneut mit der App interagieren.

Abrufen der Ergebnisse einer Fernzugriffssitzung in AWS Device Farm

Weitere Informationen zu Sitzungen finden Sie unter Sitzungen.

- Voraussetzungen
- Sitzungsdetails anzeigen
- Sitzungsvideos oder Sitzungsprotokolle werden heruntergeladen

Voraussetzungen

- Führen Sie eine Sitzung durch. Befolgen Sie die Anweisungen unter Verwenden einer Fernzugriffssitzung in AWS Device Farm, und kehren Sie dann zu dieser Seite zurück.

Sitzungsdetails anzeigen

Wenn eine Fernzugriffssitzung endet, zeigt die Device Farm Farm-Konsole eine Tabelle mit Details zu den Aktivitäten während der Sitzung an. Weitere Informationen finden Sie unter Analysieren von Protokollinformationen.

Gehen Sie wie folgt vor, um die Details einer Sitzung zu einem späteren Zeitpunkt erneut anzuzeigen:

1. Wählen Sie im Navigationsbereich Device Farm die Option Mobile Device Testing und dann Projects aus.
2. Wählen Sie das Projekt aus, das die Sitzung enthält.
3. Wählen Sie Fernzugriff und wählen Sie dann die Sitzung, die Sie überprüfen möchten, aus der Liste aus.

Sitzungsvideos oder Sitzungsprotokolle werden heruntergeladen

Wenn eine Fernzugriffssitzung endet, bietet die Device Farm Farm-Konsole Zugriff auf eine Videoaufzeichnung der Sitzungs- und Aktivitätsprotokolle. Wählen Sie in den Sitzungsergebnissen die Registerkarte Files (Dateien), um eine Liste mit Links zu den Sitzungsvideos und -protokollen anzuzeigen. Sie können diese Dateien im Browser anzeigen oder lokal speichern.

Appium-Tests in der AWS Device Farm

Während einer Fernzugriffssitzung können Sie Appium-Tests von Ihrer lokalen Umgebung aus ausführen und dabei über einen verwalteten Appium-Endpunkt auf das Gerät der Sitzung abzielen. Mit einem Appium-Endpunkt können Sie Appium-Code mit schnellem Feedback und schneller Iteration entwickeln, testen und ausführen. Dieser clientseitige Testansatz bietet die Flexibilität, von einer beliebigen Appium-Client-Umgebung Ihrer Wahl aus eine Verbindung zu einem Device Farm Farm-Gerät herzustellen.

Als Ergänzung zu clientseitigen Tests unterstützt Device Farm auch das Ausführen von Tests auf der vom Dienst verwalteten Infrastruktur, die als serverseitige Ausführung bezeichnet wird. Bei diesem Ansatz können Sie Ihre App und Tests in den Service hochladen und die Tests dann parallel auf mehreren Geräten mithilfe von vom Service verwalteten Testhosts ausführen. Dieser Ansatz eignet sich gut für Tests auf vielen Geräten unabhängig voneinander sowie für Tests im Kontext einer CI/CD Pipeline.

Weitere Informationen zur serverseitigen Ausführung finden Sie unterTest-Frameworks und integrierte Tests.

Themen

- Was ist ein Appium-Endpunkt?
- Erste Schritte mit Appium-Tests
- Interaktion mit dem Gerät mithilfe von Appium
- Überprüfung Ihrer Appium-Serverprotokolle
- Unterstützte Appium-Funktionen und -Befehle

Was ist ein Appium-Endpunkt?

Appium ist ein beliebtes Open-Source-Software-Test-Framework zum Testen nativer, hybrider und mobiler Webanwendungen auf verschiedenen Geräten, einschließlich Mobiltelefonen und Tablets, sowohl für iOS als auch für Android. Es ermöglicht Entwicklern und QA-Technikern (Qualitätssicherung), Skripte zu schreiben, mit denen ein Gerät ferngesteuert, Benutzerinteraktionen simuliert und überprüft werden kann, ob sich die zu testende Anwendung erwartungsgemäß verhält. Appium interagiert mit Apps aus der Perspektive eines Endbenutzers und ermöglicht es Testern, Tests zu entwickeln, die simulieren, wie echte Benutzer die App für ihre Tests verwenden werden.

Appium basiert auf dem Client-Server-Modell, bei dem ein lokaler Client einen (lokalen oder entfernten) Appium-Server anfordert, ein Gerät in seinem Namen zu steuern. Der Appium-Server verwaltet einen Treiber für die Kommunikation mit dem Gerät, z. B. den UIAutomator2 Treiber für Android oder den XCUITest Treiber für iOS. Alle Befehle folgen den WebDriverW3C-Standards für die Steuerung eines Geräts.

Der Appium-Endpunkt von Device Farm macht eine Appium-Server-URL für das Gerät in Ihrer Fernzugriffssitzung verfügbar. Die Appium-Endpunkt-URL ist für dieses Gerät in dieser Sitzung spezifisch und bleibt für die Dauer der Sitzung gültig, sodass Sie ohne zusätzliche Einrichtungszeit auf demselben Gerät iterieren können. Weitere Informationen zum Fernzugriff finden Sie unter. Fernzugriff

Erste Schritte mit Appium-Tests

Für die meisten Appium-Benutzer erfordert die Verwendung von Device Farm für Appium-Tests nur geringfügige Änderungen an Ihrer vorhandenen Testkonfiguration.

Im Großen und Ganzen besteht die Verwendung von Device Farm für clientseitige Appium-Tests aus drei Schritten:

1. Zunächst müssen Sie eine Fernzugriffssitzung zum Testen eines Device Farm Farm-Geräts erstellen. Sie können Ihre Apps in Ihre Fernzugriffsanfrage aufnehmen oder Apps nach dem Start der Sitzung installieren.
2. Sobald die Sitzung läuft, können Sie die Appium-Endpunkt-URL kopieren und sie entweder über ein eigenständiges Tool (wie Appium Inspector) oder über Ihren Appium-Testcode in Ihrer IDE verwenden. Die URL ist für die Dauer der Fernzugriffssitzung gültig.
3. Und schließlich können Sie nach dem Start Ihres Appium-Tests Ihre Appium-Serverprotokolle während der Testausführung zusammen mit dem Videostream Ihres Geräts live überprüfen.

Interaktion mit dem Gerät mithilfe von Appium

Sobald Sie eine Fernzugriffssitzung erstellt haben, steht das Gerät für Appium-Tests zur Verfügung. Während der gesamten Dauer der Fernzugriffssitzung können Sie auf dem Gerät so viele Appium-Sitzungen ausführen, wie Sie möchten, ohne Einschränkungen, welche Clients Sie verwenden. Sie können beispielsweise damit beginnen, einen Test mit Ihrem lokalen Appium-Code von Ihrer IDE aus auszuführen und dann zur Verwendung von Appium Inspector übergehen, um alle auftretenden Probleme zu beheben. Die Sitzung kann bis zu 150 Minuten dauern. Wenn jedoch länger als 5

Minuten keine Aktivität stattfindet (entweder über die interaktive Konsole oder über den Appium-Endpunkt), wird die Sitzung unterbrochen.

Apps zum Testen mit Ihrer Appium-Sitzung verwenden

Device Farm ermöglicht es Ihnen, Ihre App (s) als Teil Ihrer Anfrage zur Erstellung einer Fernzugriffssitzung zu verwenden oder Apps während der Fernzugriffssitzung selbst zu installieren. Diese Apps werden automatisch auf dem zu testenden Gerät installiert und als Standardfunktionen für alle Appium-Sitzungsanfragen bereitgestellt. Wenn Sie eine Fernzugriffssitzung erstellen, haben Sie die Möglichkeit, einen App-ARN zu übergeben, der standardmäßig als `appium:app` Funktion für alle nachfolgenden Appium-Sitzungen verwendet wird, sowie eine Hilfs-App ARNs, die als `appium:otherApps` Funktion verwendet wird.

Wenn Sie beispielsweise eine Fernzugriffssitzung mit einer App `com.aws.devicefarm.sample` als App und `com.aws.devicefarm.other.sample` als einer Ihrer Zusatz-Apps erstellen, verfügt sie bei der Erstellung einer Appium-Sitzung über Funktionen, die den folgenden ähneln:

```
{
    "value":
    {
        "sessionId": "abcdef123456-1234-5678-abcd-abcdef123456",
        "capabilities":
        {
            "app": "/tmp/com.aws.devicefarm.sample.apk",
            "otherApps": "[\"/tmp/com.aws.devicefarm.other.sample.apk\"]",
            ...
        }
    }
}
```

Während Ihrer Sitzung können Sie zusätzliche Apps installieren (entweder innerhalb der Konsole oder mithilfe der `InstallToRemoteAccessSession`API). Diese überschreiben alle vorhandenen Apps, die zuvor als `appium:app` Funktion verwendet wurden. Wenn diese zuvor verwendeten Apps einen eindeutigen Paketnamen haben, bleiben sie auf dem Gerät und werden als Teil der `appium:otherApps` Funktion verwendet.

Wenn Sie beispielsweise `com.aws.devicefarm.sample` bei der Erstellung Ihrer Fernzugriffssitzung zunächst eine App verwenden, dann aber `com.aws.devicefarm.other.sample` während der Sitzung eine neue App mit dem Namen installieren, verfügen Ihre Appium-Sitzungen über Funktionen, die den folgenden ähneln:

```
{
    "value":
    {
        "sessionId": "abcdef123456-1234-5678-abcd-abcdef123456",
        "capabilities":
        {
            "app": "/tmp/com.aws.devicefarm.other.sample.apk",
            "otherApps": "[\"/tmp/com.aws.devicefarm.sample.apk\"]",
            ...
        }
    }
}
```

Wenn Sie möchten, können Sie die Funktionen für Ihre App explizit anhand des App-Namens angeben (mit den `appium:bundleId` Funktionen `appium:appPackage` oder für Android bzw. iOS).

Wenn Sie eine Web-App testen, geben Sie die `browserName` Funktion für Ihre Appium-Sitzungserstellungsanfrage an. Der `Chrome` Browser ist auf allen Android-Geräten verfügbar, und der `Safari` Browser ist auf allen iOS-Geräten verfügbar.

Device Farm unterstützt die Übergabe einer Remote-URL oder eines lokalen Dateisystempfads `appium:app` während einer Fernzugriffssitzung nicht. Laden Sie Apps auf Device Farm hoch und nehmen Sie sie stattdessen in die Sitzung auf.

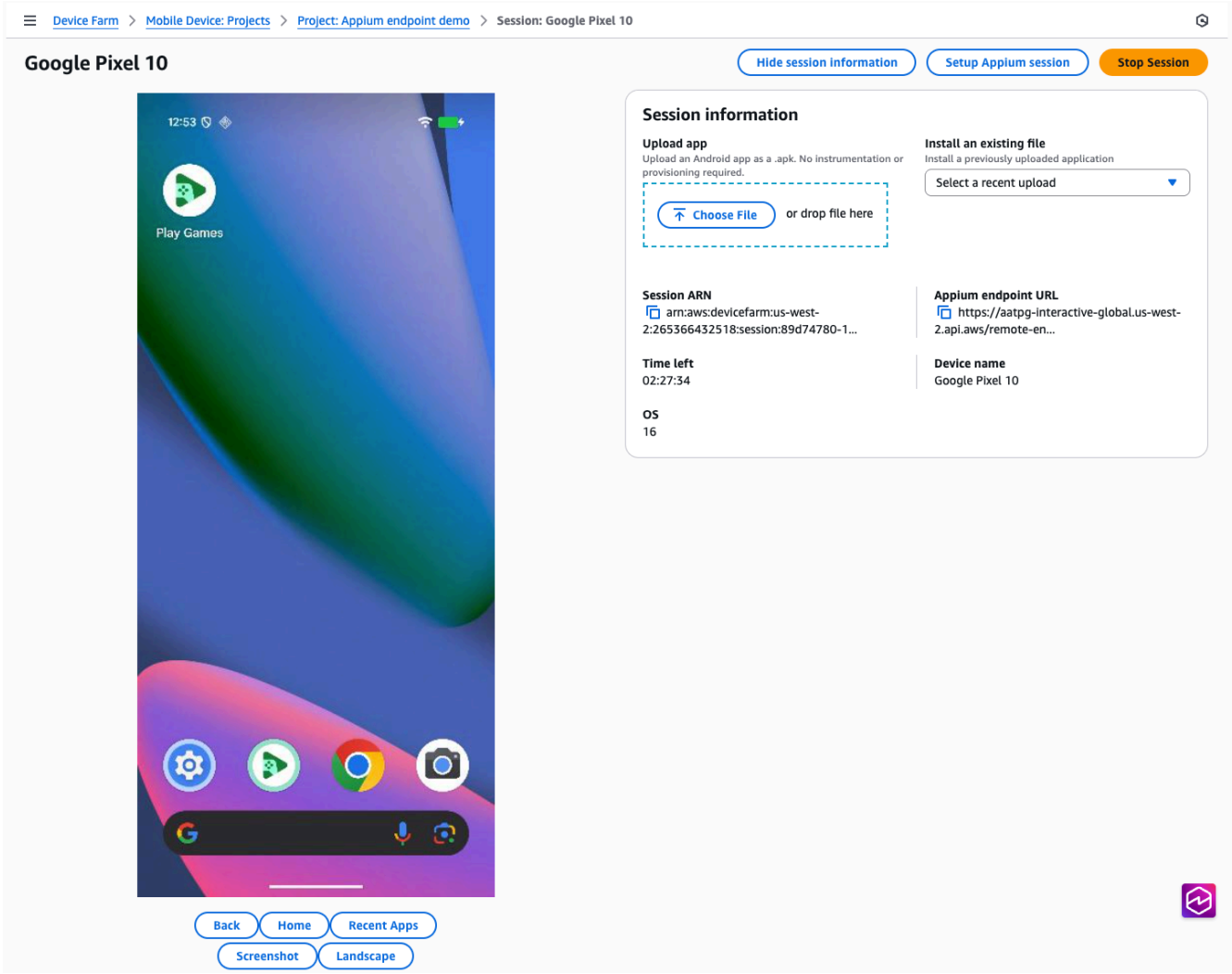
> **Note**
>
> Weitere Informationen zum automatischen Hochladen von Apps als Teil Ihrer Fernzugriffssitzung finden Sie unter Automatisieren von App-Uploads.

Wie benutzt man den Appium-Endpunkt

Hier sind die Schritte für den Zugriff auf den Appium-Endpunkt der Sitzung über die Konsole AWS CLI, den und den. AWS SDKs Diese Schritte beinhalten die ersten Schritte mit der Ausführung von Tests unter Verwendung verschiedener Appium-Client-Testframeworks:

Console

1. Öffnen Sie die Seite Ihrer Fernzugriffssitzung in Ihrem Webbrowser:



2. Gehen Sie wie folgt vor, um eine Sitzung mit Appium Inspector auszuführen:

   a. Klicken Sie auf die Schaltfläche Appium-Sitzung einrichten

   b. Folgen Sie den Anweisungen auf der Seite zum Starten einer Sitzung mit Appium Inspector.

3. Gehen Sie wie folgt vor, um einen Appium-Test von Ihrer lokalen IDE aus auszuführen:

   a. Klicken Sie auf das Symbol „Kopieren“ neben dem Text Appium-Endpunkt-URL

   b. Fügen Sie diese URL an der Stelle, an der Sie derzeit Ihre Remote-Adresse oder Ihren Befehlsausführer angeben, in Ihren lokalen Appium-Code ein. Für sprachspezifische Beispiele klicken Sie bitte auf eine der Registerkarten in diesem Beispielfenster für die Sprache Ihrer Wahl.

AWS CLI

Stellen Sie zunächst sicher, dass Ihre AWS-CLI-Version vorhanden ist, up-to-date indem Sie die neueste Version herunterladen und installieren.

> **Important**
>
> Das Appium-Endpunktfeld ist in älteren Versionen der AWS-CLI nicht verfügbar.

Sobald Ihre Sitzung läuft, ist die Appium-Endpunkt-URL über ein Feld verfügbar, das `remoteDriverEndpoint` in der Antwort auf einen API-Aufruf benannt ist: `GetRemoteAccessSession`

```
$ aws devicefarm get-remote-access-session \
    --arn "arn:aws:devicefarm:us-west-2:123456789876:session:abcdef123456-1234-5678-
abcd-abcdef123456/abcdef123456-1234-5678-abcd-abcdef123456/00000"
```

Dadurch werden Ausgaben wie die folgende angezeigt:

```
{
    "remoteAccessSession": {
        "arn": "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000",
        "name": "Google Pixel 8",
        "status": "RUNNING",
        "endpoints": {
            "remoteDriverEndpoint": "https://devicefarm-interactive-global.us-
west-2.api.aws/remote-endpoint/ABCD1234...",
        ...
}
```

Sie können diese URL in Ihrem lokalen Appium-Code überall dort verwenden, wo Sie derzeit Ihre Remote-Adresse oder Ihren Befehlsausführer angeben. Für sprachspezifische Beispiele klicken Sie bitte auf eine der Registerkarten in diesem Beispielfenster für die Sprache Ihrer Wahl.

Ein Beispiel dafür, wie Sie direkt von der Befehlszeile aus mit dem Endpunkt interagieren können, finden Sie mit dem Befehlszeilentool curl, um einen Endpunkt direkt aufzurufen: WebDriver

```
$ curl "https://devicefarm-interactive-global.us-west-2.api.aws/remote-endpoint/
ABCD1234.../status"
```

Dadurch werden Ausgaben wie die folgende angezeigt:

```
{
    "value":
    {
        "ready": true,
        "message": "The server is ready to accept new connections",
        "build":
        {
            "version": "2.5.1"
        }
    }
}
```

Python

Sobald Ihre Sitzung läuft, ist die Appium-Endpunkt-URL über ein Feld verfügbar, das `remoteDriverEndpoint` in der Antwort auf einen GetRemoteAccessSessionAPI-Aufruf benannt ist:

```
# To get the URL
import sys
import boto3
from botocore.exceptions import ClientError

def get_appium_endpoint() -> str:
    session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000"
    device_farm_client = boto3.client("devicefarm", region_name="us-west-2")

    try:
        resp = device_farm_client.get_remote_access_session(arn=session_arn)
    except ClientError as exc:
        sys.exit(f"Failed to call Device Farm: {exc}")

    remote_access_session = resp.get("remoteAccessSession", {})
    endpoints = remote_access_session.get("endpoints", {})
    endpoint = endpoints.get("remoteDriverEndpoint")
```

```
    if not endpoint:
        sys.exit("Device Farm response did not include
 endpoints.remoteDriverEndpoint")

    return endpoint

# To use the URL
from appium import webdriver
from appium.options.android import UiAutomator2Options

opts = UiAutomator2Options()
driver = webdriver.Remote(get_appium_endpoint(), options=opts)
# ...
driver.quit()
```

Java

Hinweis: Dieses Beispiel verwendet das AWS SDK for Java v2 und ist mit JDK-Versionen 11 und höher kompatibel.

Sobald Ihre Sitzung läuft, ist die Appium-Endpunkt-URL über ein Feld verfügbar, das `remoteDriverEndpoint` in der Antwort auf einen API-Aufruf benannt ist: GetRemoteAccessSession

```
// To get the URL
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import
 software.amazon.awssdk.services.devicefarm.model.GetRemoteAccessSessionRequest;
import
 software.amazon.awssdk.services.devicefarm.model.GetRemoteAccessSessionResponse;

public class AppiumEndpointBuilder {
    public static String getAppiumEndpoint() throws Exception {
        String session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000";

        try (DeviceFarmClient client = DeviceFarmClient.builder()
                .region(Region.US_WEST_2)
                .credentialsProvider(DefaultCredentialsProvider.create())
```

```
                .build()) {

            GetRemoteAccessSessionResponse resp = client.getRemoteAccessSession(
                    GetRemoteAccessSessionRequest.builder().arn(session_arn).build()
            );

            String endpoint =
 resp.remoteAccessSession().endpoints().remoteDriverEndpoint();
            if (endpoint == null || endpoint.isEmpty()) {
                throw new IllegalStateException("remoteDriverEndpoint missing from
 response");
            }
            return endpoint;
        }
    }
}

// To use the URL
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.options.UiAutomator2Options;

import java.net.URL;

public class ExampleTest {
    public static void main(String[] args) throws Exception {
        String endpoint = AppiumEndpointBuilder.getAppiumEndpoint();
        UiAutomator2Options options = new UiAutomator2Options();
        AndroidDriver driver = new AndroidDriver(new URL(endpoint), options);

        try {
            // ... your test ...
        } finally {
            driver.quit();
        }
    }
}
```

JavaScript

Hinweis: Dieses Beispiel verwendet AWS SDK für JavaScript v3 und WebDriverIO v8+ mit Node 18+.

Sobald Ihre Sitzung läuft, ist die Appium-Endpunkt-URL über ein Feld verfügbar, das `remoteDriverEndpoint` in der Antwort auf einen API-Aufruf benannt ist: [GetRemoteAccessSession]

```
// To get the URL
import { DeviceFarmClient, GetRemoteAccessSessionCommand } from "@aws-sdk/client-
device-farm";

export async function getAppiumEndpoint() {
  const sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000";

  const client = new DeviceFarmClient({ region: "us-west-2" });
  const resp = await client.send(new GetRemoteAccessSessionCommand({ arn:
 sessionArn }));

  const endpoint = resp?.remoteAccessSession?.endpoints?.remoteDriverEndpoint;
  if (!endpoint) throw new Error("remoteDriverEndpoint missing from response");
  return endpoint;
}

// To use the URL with WebdriverIO
import { remote } from "webdriverio";

(async () => {
  const endpoint = await getAppiumEndpoint();
  const u = new URL(endpoint);

  const driver = await remote({
    protocol: u.protocol.replace(":", ""),
    hostname: u.hostname,
    port: u.port ? Number(u.port) : (u.protocol === "https:" ? 443 : 80),
    path: u.pathname + u.search,
    capabilities: {
      platformName: "Android",
      "appium:automationName": "UiAutomator2",
      // ...other caps...
    },
  });

  try {
    // ... your test ...
```

```
  } finally {
    await driver.deleteSession();
  }
})();
```

C#

Sobald Ihre Sitzung läuft, ist die Appium-Endpunkt-URL über ein Feld verfügbar, das `remoteDriverEndpoint` in der Antwort auf einen API-Aufruf benannt ist: [GetRemoteAccessSession]

```
// To get the URL
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

public static class AppiumEndpointBuilder
{
    public static async Task<string> GetAppiumEndpointAsync()
    {
        var sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000";

        var config = new AmazonDeviceFarmConfig
        {
            RegionEndpoint = RegionEndpoint.USWest2
        };
        using var client = new AmazonDeviceFarmClient(config);

        var resp = await client.GetRemoteAccessSessionAsync(new
 GetRemoteAccessSessionRequest { Arn = sessionArn });
        var endpoint = resp?.RemoteAccessSession?.Endpoints?.RemoteDriverEndpoint;

        if (string.IsNullOrWhiteSpace(endpoint))
            throw new InvalidOperationException("RemoteDriverEndpoint missing from
 response");

        return endpoint;
    }
}
```

```
// To use the URL
using OpenQA.Selenium.Appium;
using OpenQA.Selenium.Appium.Android;

class Example
{
    static async Task Main()
    {
        var endpoint = await AppiumEndpointBuilder.GetAppiumEndpointAsync();

        var options = new AppiumOptions();
        options.PlatformName = "Android";
        options.AutomationName = "UiAutomator2";

        using var driver = new AndroidDriver(new Uri(endpoint), options);
        try
        {
            // ... your test ...
        }
        finally
        {
            driver.Quit();
        }
    }
}
```

Ruby

Sobald Ihre Sitzung läuft, ist die Appium-Endpunkt-URL über ein Feld verfügbar, das `remoteDriverEndpoint` in der Antwort auf einen API-Aufruf benannt ist: GetRemoteAccessSession

```
# To get the URL
require 'aws-sdk-devicefarm'

def get_appium_endpoint
  session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000"

  client = Aws::DeviceFarm::Client.new(region: 'us-west-2')
  resp = client.get_remote_access_session(arn: session_arn)
```

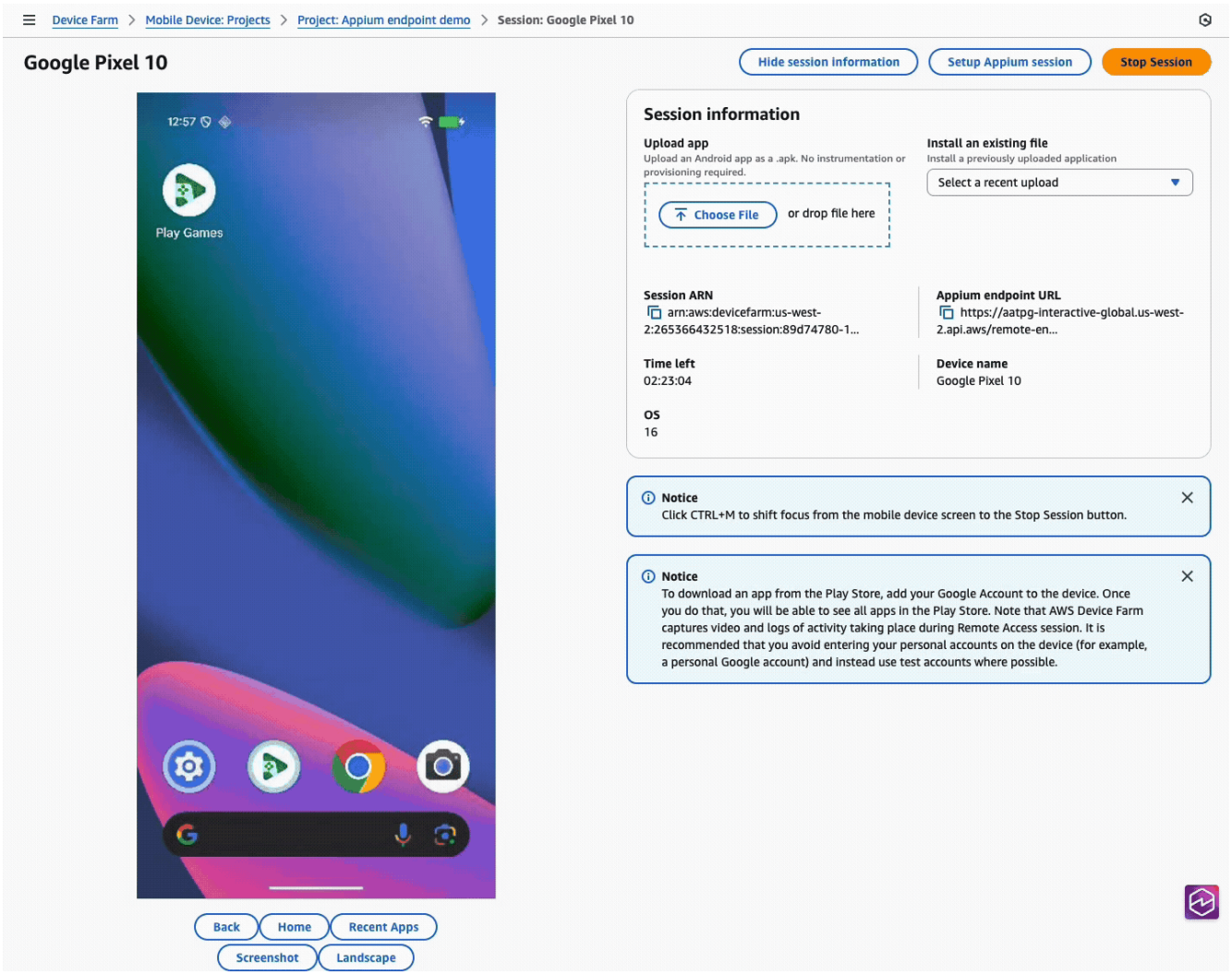
```
  endpoint = resp.remote_access_session.endpoints.remote_driver_endpoint
  raise "remote_driver_endpoint missing from response" if endpoint.nil? ||
 endpoint.empty?
  endpoint
end

# To use the URL
require 'appium_lib_core'

endpoint = get_appium_endpoint
opts = {
  server_url: endpoint,
  capabilities: {
    'platformName' => 'Android',
    'appium:automationName' => 'UiAutomator2'
  }
}

driver = Appium::Core.for(opts).start_driver
begin
  # ... your test ...
ensure
  driver.quit
end
```

Überprüfung Ihrer Appium-Serverprotokolle

Sobald Sie eine Appium-Sitzung gestartet haben, können Sie die Appium-Serverprotokolle live in der Device Farm Farm-Konsole anzeigen oder sie nach dem Ende der Fernzugriffssitzung herunterladen. Hier sind die Anweisungen dazu:

Console

1. Öffnen Sie in der Device Farm Farm-Konsole die Fernzugriffssitzung für Ihr Gerät.
2. Starten Sie eine Appium-Endpunktsitzung mit dem Gerät von Ihrer lokalen IDE oder Appium Inspector aus
3. Anschließend wird das Appium-Serverprotokoll neben dem Gerät auf der Seite für die Fernzugriffssitzung angezeigt. Die „Sitzungsinformationen“ finden Sie unten auf der Seite unter dem Gerät:



AWS CLI

Hinweis: In diesem Beispiel wird das Befehlszeilentool verwendet`curl`, um das Protokoll von Device Farm abzurufen.

Während oder nach der Sitzung können Sie die `ListArtifacts`API von Device Farm verwenden, um das Appium-Serverprotokoll herunterzuladen.

```
$ aws devicefarm list-artifacts \
  --type FILE \
  --arn arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678
```

Dadurch werden während der Sitzung Ausgaben wie die folgende angezeigt:

```
{
    "artifacts": [
        {
            "arn": "arn:aws:devicefarm:us-
west-2:111122223333:artifact:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-4567
            "name": "AppiumServerLogOutput",
            "type": "APPIUM_SERVER_LOG_OUTPUT",
            "extension": "",
            "url": "https://prod-us-west-2-results.s3.dualstack.us-
west-2.amazonaws.com/111122223333/12345678..."
        }
    ]
}
```

Und das Folgende, nachdem die Sitzung abgeschlossen ist:

```
{
    "artifacts": [
        {
            "arn": "arn:aws:devicefarm:us-
west-2:111122223333:artifact:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-4567
            "name": "Appium Server Output",
            "type": "APPIUM_SERVER_OUTPUT",
            "extension": "log",
            "url": "https://prod-us-west-2-results.s3.dualstack.us-
west-2.amazonaws.com/111122223333/12345678..."
        }
    ]
}
```

```
$ curl "https://prod-us-west-2-results.s3.dualstack.us-
west-2.amazonaws.com/111122223333/12345678..."
```

Dadurch werden Ausgaben wie die folgende angezeigt:

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1',
info Appium   allowInsecure:
info Appium    [ 'execute_driver_script',
info Appium      'session_discovery',
info Appium      'perf_record',
```

```
info Appium      'adb_shell',
info Appium      'chromedriver_autodownload',
info Appium      'get_server_logs' ],
info Appium   keepAliveTimeout: 0,
info Appium   logNoColors: true,
info Appium   logTimestamp: true,
info Appium   longStacktrace: true,
info Appium   sessionOverride: true,
info Appium   strictCaps: true,
info Appium   useDrivers: [ 'uiautomator' ] }
```

Python

Hinweis: In diesem Beispiel werden das *`requests`* Drittanbieterpaket zum Herunterladen des Protokolls sowie das AWS SDK für Python verwendet*`boto3`*.

Während oder nach der Sitzung können Sie die [ListArtifacts](#)API von Device Farm verwenden, um die URL des Appium-Serverprotokolls abzurufen und sie dann herunterzuladen.

```
import pathlib
import requests
import boto3

def download_appium_log():
    session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678
    client = boto3.client("devicefarm", region_name="us-west-2")

    # 1) List artifacts for the session (FILE artifacts), handling pagination
    artifacts = []
    token = None
    while True:
        kwargs = {"arn": session_arn, "type": "FILE"}
        if token:
            kwargs["nextToken"] = token
        resp = client.list_artifacts(**kwargs)
        artifacts.extend(resp.get("artifacts", []))
        token = resp.get("nextToken")
        if not token:
            break

    if not artifacts:
        raise RuntimeError("No artifacts found in this session")
```

```
    # Filter strictly to Appium server logs
    allowed = {"APPIUM_SERVER_OUTPUT", "APPIUM_SERVER_LOG_OUTPUT"}
    filtered = [a for a in artifacts if a.get("type") in allowed]
    if not filtered:
        raise RuntimeError("No Appium server log artifacts found (expected
 APPIUM_SERVER_OUTPUT or APPIUM_SERVER_LOG_OUTPUT)")

    # Prefer the final 'OUTPUT' log, else the live 'LOG_OUTPUT'
    chosen = (next((a for a in filtered if a.get("type") == "APPIUM_SERVER_OUTPUT"),
 None)
              or next((a for a in filtered if a.get("type") ==
 "APPIUM_SERVER_LOG_OUTPUT"), None))

    url = chosen["url"]
    ext = chosen.get("extension") or "log"
    out = pathlib.Path(f"./appium_server_log.{ext}")

    # 2) Download the artifact
    with requests.get(url, stream=True) as r:
        r.raise_for_status()
        with open(out, "wb") as fh:
            for chunk in r.iter_content(chunk_size=1024 * 1024):
                if chunk:
                    fh.write(chunk)

    print(f"Saved Appium server log to: {out.resolve()}")

download_appium_log()
```

Dadurch werden Ausgaben wie die folgende angezeigt:

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', allowInsecure: [ 'execute_driver_script', ... ],
 useDrivers: [ 'uiautomator' ] }
```

Java

Hinweis: Dieses Beispiel verwendet das AWS SDK for Java v2 und *`HttpClient`* zum Herunterladen des Protokolls und ist mit JDK-Versionen 11 und höher kompatibel.

Während oder nach der Sitzung können Sie die ListArtifactsAPI von Device Farm verwenden, um die URL des Appium-Serverprotokolls abzurufen und sie dann herunterzuladen.

```
import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.nio.file.Path;
import java.time.Duration;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import software.amazon.awssdk.services.devicefarm.model.Artifact;
import software.amazon.awssdk.services.devicefarm.model.ArtifactCategory;
import software.amazon.awssdk.services.devicefarm.model.ListArtifactsRequest;
import software.amazon.awssdk.services.devicefarm.model.ListArtifactsResponse;

public class AppiumLogDownloader {

    public static void main(String[] args) throws Exception {
        String sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678

        try (DeviceFarmClient client = DeviceFarmClient.builder()
                .region(Region.US_WEST_2)
                .build()) {

            // 1) List artifacts for the session (FILE artifacts) with pagination
            List<Artifact> all = new ArrayList<>();
            String token = null;
            do {
                ListArtifactsRequest.Builder b = ListArtifactsRequest.builder()
                        .arn(sessionArn)
                        .type(ArtifactCategory.FILE);
                if (token != null) b.nextToken(token);
                ListArtifactsResponse page = client.listArtifacts(b.build());
                all.addAll(page.artifacts());
                token = page.nextToken();
            } while (token != null && !token.isBlank());
```

```
            // Filter strictly to Appium logs
            List<Artifact> filtered = all.stream()
                    .filter(a -> {
                        String t = a.typeAsString();
                        return "APPIUM_SERVER_OUTPUT".equals(t) ||
"APPIUM_SERVER_LOG_OUTPUT".equals(t);
                    })
                    .toList();

            if (filtered.isEmpty()) {
                throw new RuntimeException("No Appium server log artifacts found
(expected APPIUM_SERVER_OUTPUT or APPIUM_SERVER_LOG_OUTPUT).");
            }

            // Prefer OUTPUT; else LOG_OUTPUT
            Artifact chosen = filtered.stream()
                    .filter(a -> "APPIUM_SERVER_OUTPUT".equals(a.typeAsString()))
                    .findFirst()
                    .orElseGet(() -> filtered.stream()
                            .filter(a ->
"APPIUM_SERVER_LOG_OUTPUT".equals(a.typeAsString()))
                            .findFirst()
                            .get());

            String url = chosen.url();
            String ext = (chosen.extension() == null ||
chosen.extension().isBlank()) ? "log" : chosen.extension();
            Path out = Path.of("appium_server_log." + ext);

            // 2) Download the artifact with HttpClient
            HttpClient http = HttpClient.newBuilder()
                    .connectTimeout(Duration.ofSeconds(10))
                    .build();

            HttpRequest get = HttpRequest.newBuilder(URI.create(url))
                    .timeout(Duration.ofMinutes(5))
                    .GET()
                    .build();

            HttpResponse<Path> resp = http.send(get,
HttpResponse.BodyHandlers.ofFile(out));
            if (resp.statusCode() / 100 != 2) {
```

```
                throw new IOException("Failed to download log, HTTP " +
 resp.statusCode());
            }
            System.out.println("Saved Appium server log to: " +
 out.toAbsolutePath());
        }
    }
}
```

Dadurch werden Ausgaben wie die folgende angezeigt:

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', ..., useDrivers: [ 'uiautomator' ] }
```

JavaScript

Hinweis: In diesem Beispiel wird AWS SDK für JavaScript (v3) und Node 18+ verwendet*fetch*, um das Protokoll herunterzuladen.

Während oder nach der Sitzung können Sie die `ListArtifacts`API von Device Farm verwenden, um die URL des Appium-Serverprotokolls abzurufen und sie dann herunterzuladen.

```
import { DeviceFarmClient, ListArtifactsCommand } from "@aws-sdk/client-device-
farm";
import { createWriteStream } from "fs";
import { pipeline } from "stream";
import { promisify } from "util";

const pipe = promisify(pipeline);
const client = new DeviceFarmClient({ region: "us-west-2" });

const sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678

// 1) List artifacts for the session (FILE artifacts), handling pagination
const artifacts = [];
let nextToken;
do {
  const page = await client.send(new ListArtifactsCommand({
    arn: sessionArn,
    type: "FILE",
```

```
    nextToken
  }));
  artifacts.push(...(page.artifacts ?? []));
  nextToken = page.nextToken;
} while (nextToken);

if (!artifacts.length) throw new Error("No artifacts found");

// Strict filter to Appium logs
const filtered = (artifacts ?? []).filter(a =>
  a.type === "APPIUM_SERVER_OUTPUT" || a.type === "APPIUM_SERVER_LOG_OUTPUT"
);
if (!filtered.length) {
  throw new Error("No Appium server log artifacts found (expected
 APPIUM_SERVER_OUTPUT or APPIUM_SERVER_LOG_OUTPUT).");
}

// Prefer OUTPUT; else LOG_OUTPUT
const chosen =
  filtered.find(a => a.type === "APPIUM_SERVER_OUTPUT") ??
  filtered.find(a => a.type === "APPIUM_SERVER_LOG_OUTPUT");

const url = chosen.url;
const ext = chosen.extension || "log";
const outPath = `./appium_server_log.${ext}`;

// 2) Download the artifact
const resp = await fetch(url);
if (!resp.ok) {
  throw new Error(`Failed to download log: ${resp.status} ${await
 resp.text().catch(()=>"")}`);
}
await pipe(resp.body, createWriteStream(outPath));
console.log("Saved Appium server log to:", outPath);
```

Dadurch werden Ausgaben wie die folgende angezeigt:

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', allowInsecure: [ 'execute_driver_script', ... ],
 useDrivers: [ 'uiautomator' ] }
```

C#

Hinweis: In diesem Beispiel wird das AWS SDK for .NET und *`HttpClient`* zum Herunterladen des Protokolls verwendet.

Während oder nach der Sitzung können Sie die `ListArtifacts`API von Device Farm verwenden, um die URL des Appium-Serverprotokolls abzurufen und sie dann herunterzuladen.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Net.Http;
using System.Threading.Tasks;
using System.Linq;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

class AppiumLogDownloader
{
    static async Task Main()
    {
        var sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678

        using var client = new AmazonDeviceFarmClient(RegionEndpoint.USWest2);

        // 1) List artifacts for the session (FILE artifacts), handling pagination
        var all = new List<Artifact>();
        string? token = null;
        do
        {
            var page = await client.ListArtifactsAsync(new ListArtifactsRequest
            {
                Arn = sessionArn,
                Type = ArtifactCategory.FILE,
                NextToken = token
            });
            if (page.Artifacts != null) all.AddRange(page.Artifacts);
            token = page.NextToken;
        } while (!string.IsNullOrEmpty(token));

        if (all.Count == 0)
```

```
            throw new Exception("No artifacts found");

        // Strict filter to Appium logs
        var filtered = all.Where(a =>
            a.Type == "APPIUM_SERVER_OUTPUT" || a.Type ==
 "APPIUM_SERVER_LOG_OUTPUT").ToList();

        if (filtered.Count == 0)
            throw new Exception("No Appium server log artifacts found (expected
 APPIUM_SERVER_OUTPUT or APPIUM_SERVER_LOG_OUTPUT).");

        // Prefer OUTPUT; else LOG_OUTPUT
        var chosen = filtered.FirstOrDefault(a => a.Type == "APPIUM_SERVER_OUTPUT")
                    ?? filtered.First(a => a.Type == "APPIUM_SERVER_LOG_OUTPUT");

        var url = chosen.Url;
        var ext = string.IsNullOrWhiteSpace(chosen.Extension) ? "log" :
 chosen.Extension;
        var outPath = $"./appium_server_log.{ext}";

        // 2) Download the artifact
        using var http = new HttpClient();
        using var resp = await http.GetAsync(url,
 HttpCompletionOption.ResponseHeadersRead);
        resp.EnsureSuccessStatusCode();
        await using (var fs = File.Create(outPath))
        {
            await resp.Content.CopyToAsync(fs);
        }
        Console.WriteLine($"Saved Appium server log to:
 {Path.GetFullPath(outPath)}");
    }
}
```

Dadurch werden Ausgaben wie die folgende angezeigt:

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', ..., useDrivers: [ 'uiautomator' ] }
```

Ruby

Hinweis: In diesem Beispiel wird das AWS SDK for Ruby und *`Net::HTTP`* zum Herunterladen des Protokolls verwendet.

Während oder nach der Sitzung können Sie die `ListArtifacts`API von Device Farm verwenden, um die URL des Appium-Serverprotokolls abzurufen und sie dann herunterzuladen.

```
require "aws-sdk-devicefarm"
require "net/http"
require "uri"

client = Aws::DeviceFarm::Client.new(region: "us-west-2")
session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678

# 1) List artifacts for the session (FILE artifacts), handling pagination
artifacts = []
token = nil
loop do
  page = client.list_artifacts(arn: session_arn, type: "FILE", next_token: token)
  artifacts.concat(page.artifacts || [])
  token = page.next_token
  break if token.nil? || token.empty?
end

raise "No artifacts found" if artifacts.empty?

# Strict filter to Appium logs
filtered = (artifacts || []).select { |a| ["APPIUM_SERVER_OUTPUT",
 "APPIUM_SERVER_LOG_OUTPUT"].include?(a.type) }
raise "No Appium server log artifacts found (expected APPIUM_SERVER_OUTPUT or
 APPIUM_SERVER_LOG_OUTPUT)." if filtered.empty?

# Prefer OUTPUT; else LOG_OUTPUT
chosen = filtered.find { |a| a.type == "APPIUM_SERVER_OUTPUT" } ||
         filtered.find { |a| a.type == "APPIUM_SERVER_LOG_OUTPUT" }

url = chosen.url
ext = (chosen.extension && !chosen.extension.empty?) ? chosen.extension : "log"
out_path = "./appium_server_log.#{ext}"

# 2) Download the artifact
```

```
uri = URI.parse(url)
Net::HTTP.start(uri.host, uri.port, use_ssl: (uri.scheme == "https")) do |http|
  req = Net::HTTP::Get.new(uri)
  http.request(req) do |resp|
    raise "Failed GET: #{resp.code} #{resp.body}" unless resp.code.to_i / 100 == 2
    File.open(out_path, "wb") { |f| resp.read_body { |chunk| f.write(chunk) } }
  end
end
puts "Saved Appium server log to: #{File.expand_path(out_path)}"
```

Dadurch werden Ausgaben wie die folgende angezeigt:

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', allowInsecure: [ 'execute_driver_script', ... ],
 useDrivers: [ 'uiautomator' ] }
```

Unterstützte Appium-Funktionen und -Befehle

Der Appium-Endpunkt von Device Farm unterstützt die meisten Befehle und gewünschten Funktionen, die Sie auf lokalen Geräten verwenden, mit wenigen Ausnahmen. Die folgenden Listen zeigen, welche Funktionen und Befehle derzeit nicht unterstützt werden. Wenn Ihre Tests aufgrund eingeschränkter Funktionen nicht wie erwartet ausgeführt werden können, wenden Sie sich bitte an einen Support-Fall, um weitere Informationen zu erhalten.

Unterstützte Funktionen

Wenn Sie eine Appium-Sitzung auf Device Farm erstellen, empfehlen wir, über einen bestimmten Funktionsumfang zu verfügen, der alle für Ihr lokales Gerät spezifischen Funktionen ausschließt. In Device Farm schlägt die Sitzungserstellung möglicherweise fehl, wenn bestimmte nicht unterstützte Funktionen eingestellt sind. Dazu gehören gerätespezifische Funktionen wie und. `udid` `platformVersion` Darüber hinaus werden bestimmte Funktionen, die sich ChromeDriver auf Android und WebDriverAgent iOS beziehen, nicht unterstützt, ebenso wie Funktionen, die nur auf Emulatoren und Simulatoren unterstützt werden.

Unterstützte Befehle

Die meisten Appium-Befehle, die auf echten Android- und iOS-Geräten ordnungsgemäß ausgeführt werden, werden wie erwartet auf Device Farm ausgeführt, mit den folgenden Ausnahmen:

Appium-Gerätebefehle () **/appium/device**

- install_app
- finger_print
- send_sms
- gsm_call
- gsm_signal
- gsm_voice
- power_ac
- power_capacity
- network_speed
- shake

Appium führt Methoden und Skripte aus () **/execute**

- installApp
- execEmuConsoleCommand
- fingerprint
- gsmCall
- gsmSignal
- sendSms
- gsmVoice
- powerAC
- powerCapacity
- networkSpeed
- sensorSet
- injectEmulatorCameraImage
- isGpsEnabled
- shake
- clearApp
- clearKeychains

- configureLocalization
- enrollBiometric
- getPasteboard
- installXCTestBundle
- listXCTestBundles
- listXCTestsInTestBundle
- runXCTest
- sendBiometricMatch
- setPasteboard
- setPermission
- startAudioRecording
- startLogsBroadcast
- startRecordingScreen
- startScreenStreaming
- startXCTestScreenRecording
- stopAudioRecording
- stopLogsBroadcast
- stopRecordingScreen
- stopScreenStreaming
- stopXCTestScreenRecording
- updateSafariPreferences

Private Geräte in AWS Device Farm

Ein privates Gerät ist ein physisches Mobilgerät, das AWS Device Farm in Ihrem Namen in einem Amazon-Rechenzentrum bereitstellt. Dieses Gerät ist exklusiv für Ihr AWS Konto verfügbar.

> **Note**
>
> Derzeit sind private Geräte nur in der Region AWS USA West (Oregon) verfügbar (`us-west-2`).

Sobald Sie eine private Geräteflotte haben, können Sie Remote-Zugriffssitzungen erstellen oder unter Verwendung Ihrer privaten Geräte Testläufe planen. Weitere Informationen finden Sie unter Einen Testlauf erstellen oder eine Fernzugriffssitzung in AWS Device Farm starten. Sie können auch Instance-Profile erstellen, um das Verhalten von privaten Geräten während eines Testlaufs oder einer Remote-Zugriffssitzung zu steuern. Weitere Informationen finden Sie unter Erstellen eines Instanzprofils in AWS Device Farm. Optional können Sie beantragen, dass bestimmte private Android-Geräte als gerootete Geräte bereitgestellt werden.

Sie können auch einen Amazon Virtual Private Cloud Cloud-Endpunktservice erstellen, um private Apps zu testen, auf die Ihr Unternehmen Zugriff hat, die aber nicht über das Internet erreichbar sind. Beispielsweise könnten Sie eine Webanwendung innerhalb Ihrer VPC ausführen, die Sie auf mobilen Geräten testen möchten. Weitere Informationen finden Sie unter Verwendung von Amazon VPC-Endpunktservices mit Device Farm — Legacy (nicht empfohlen).

Wenn Sie daran interessiert sind, eine Flotte von privaten Geräten zu verwenden, kontaktieren Sie uns. Das Device Farm Farm-Team muss mit Ihnen zusammenarbeiten, um eine Flotte von privaten Geräten für Ihr AWS Konto einzurichten und bereitzustellen.

Themen

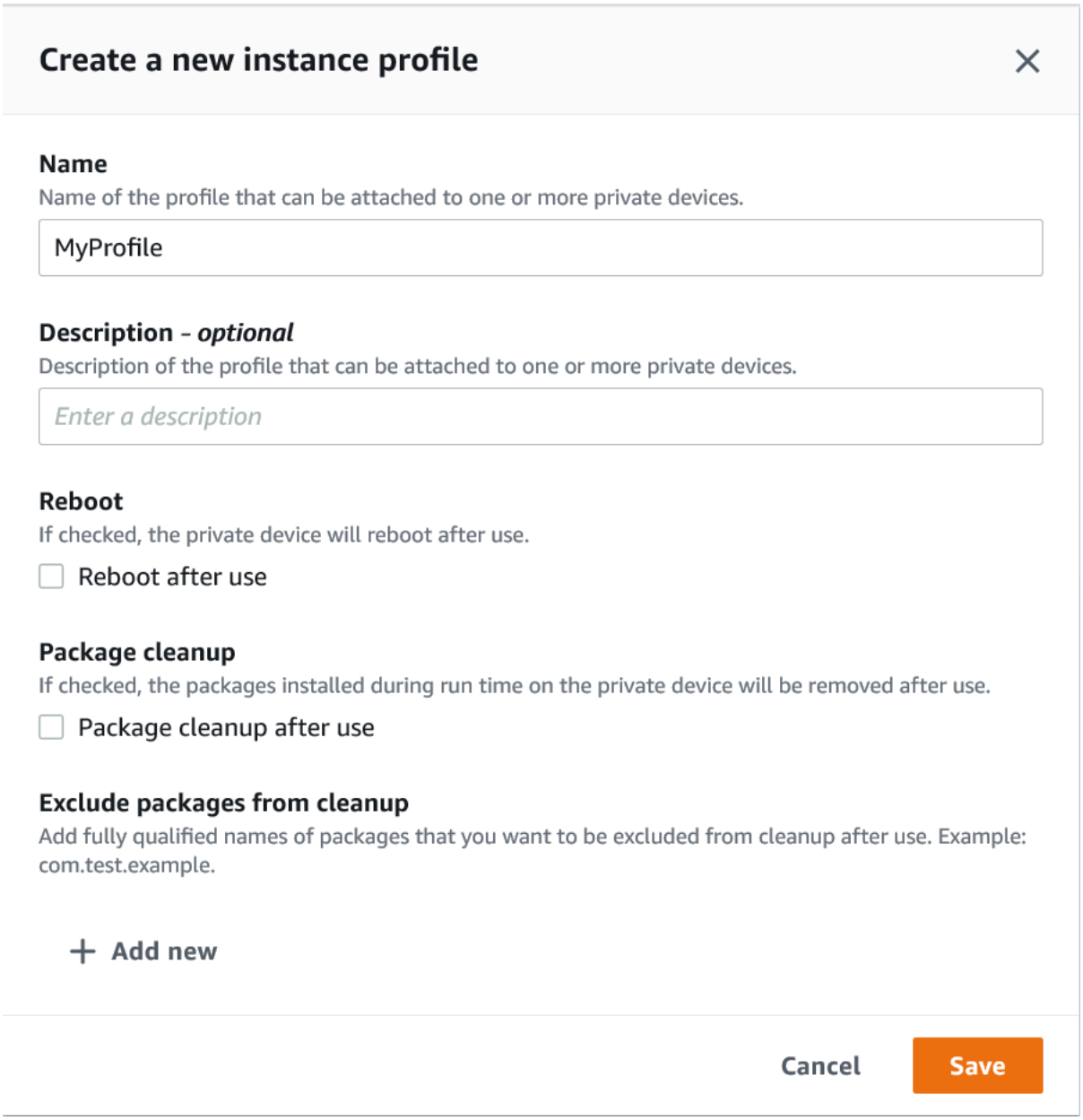
- Private Geräte in Device Farm beenden

Erstellen eines Instanzprofils in AWS Device Farm

Sie können eine Flotte mit einem oder mehreren privaten Geräten einrichten. Diese Geräte sind Ihrem AWS -Konto zugeordnet. Nachdem Sie die Geräte eingerichtet haben, können Sie optional ein oder mehrere Instance-Profile erstellen. Mithilfe der Instance-Profile werden die Testläufe automatisiert und auf den Gerät-Instances durchgängig dieselben Einstellungen angewendet. Instanzprofile können Ihnen auch dabei helfen, das Verhalten von Fernzugriffssitzungen zu kontrollieren. Weitere Informationen zu privaten Geräten in Device Farm finden Sie unterPrivate Geräte in AWS Device Farm.

So erstellen Sie eine -Instance

1. Öffnen Sie die Device Farm Farm-Konsole unter https://console.aws.amazon.com/devicefarm/.
2. Wählen Sie im Navigationsbereich Device Farm die Option Mobile Device Testing und dann Private Geräte aus.
3. Wählen Sie Instance profiles (Instance-Profile) aus.
4. Wählen Sie Instance-Profil erstellen aus.
5. Geben Sie einen Namen für das Instance-Profil ein.



6. (Optional) Geben Sie eine Beschreibung für das Instance-Profil ein.
7. (Optional) Ändern Sie eine der folgenden Einstellungen, um anzugeben, welche Aktionen Device Farm nach dem Ende jedes Testlaufs oder jeder Sitzung auf einem Gerät ausführen soll:

   - Nach Gebrauch neu starten — Um das Gerät neu zu starten, aktivieren Sie dieses Kontrollkästchen. Das Kontrollkästchen ist standardmäßig deaktiviert (`false`).
   - Paketbereinigung — Um alle App-Pakete zu entfernen, die Sie auf dem Gerät installiert haben, aktivieren Sie dieses Kontrollkästchen. Das Kontrollkästchen ist standardmäßig deaktiviert (`false`). Wenn Sie alle App-Pakete, die Sie auf dem Gerät installiert haben, beibehalten möchten, lassen Sie das Kontrollkästchen deaktiviert.

- Pakete von der Säuberung ausschließen — Um nur ausgewählte App-Pakete auf dem Gerät zu behalten, aktivieren Sie das Kontrollkästchen Paketbereinigung und wählen Sie dann Neu hinzufügen. Geben Sie als Paketnamen den vollständig qualifizierten Namen des App-Pakets ein, das Sie auf dem Gerät behalten möchten (z. B. `com.test.example`). Wählen Sie Add new (Neue hinzufügen) aus, wenn Sie mehrere App-Pakete auf dem Gerät beibehalten möchten. Geben Sie anschließend den vollständig qualifizierten Namen der einzelnen Pakete ein.

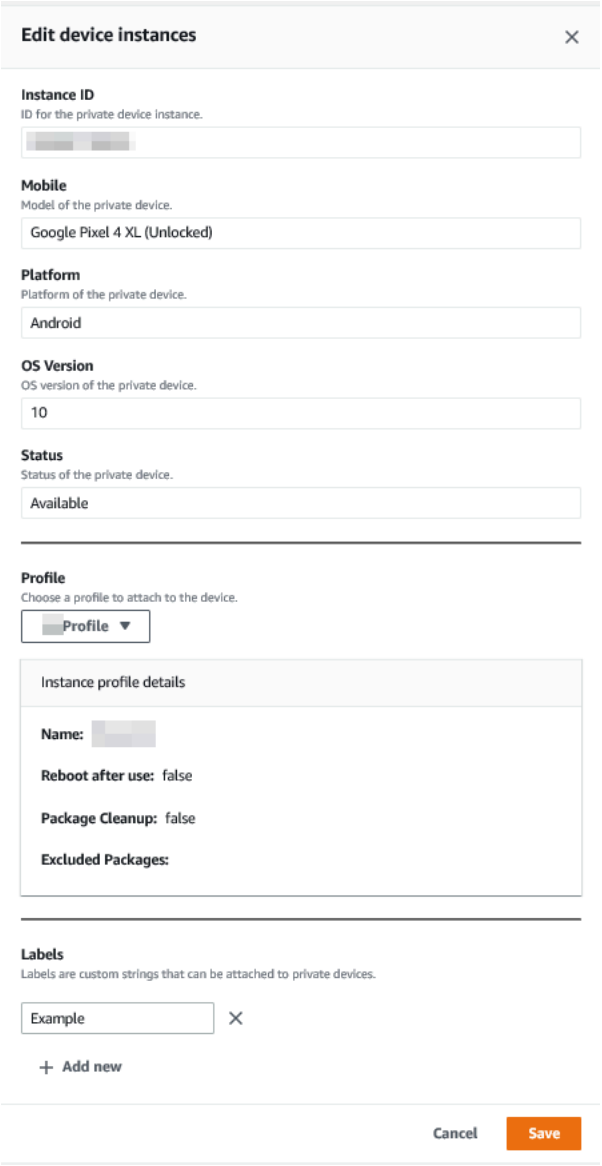
8. Wählen Sie Speichern.

Zusätzliche private Geräte in AWS Device Farm anfordern

In AWS Device Farm können Sie beantragen, dass zusätzliche private Geräteinstanzen zu Ihrer Flotte hinzugefügt werden. Sie können auch die Einstellungen vorhandener privater Geräteinstanzen in Ihrer Flotte anzeigen und ändern. Weitere Informationen zu privaten Geräten finden Sie unterPrivate Geräte in AWS Device Farm.

Um zusätzliche private Geräte anzufordern oder deren Einstellungen zu ändern

1. Öffnen Sie die Device Farm Farm-Konsole unter https://console.aws.amazon.com/devicefarm/.
2. Wählen Sie im Navigationsbereich Device Farm die Option Mobile Device Testing und dann Private Geräte aus.
3. Wählen Sie Device Instances (Geräte-Instances) aus. Die Registerkarte Device instances (Geräte-Instances) zeigt Ihnen eine Tabelle mit den privaten Geräten Ihrer Flotte an. Um die Tabelle schnell zu durchsuchen oder zu filtern, geben Sie Suchbegriffe in die Suchleiste über den Spalten ein.
4. Um eine neue private Geräteinstanz anzufordern, wählen Sie Geräteinstanz anfordern oder kontaktieren Sie uns. Private Geräte erfordern eine zusätzliche Einrichtung mit Hilfe des Device Farm Farm-Teams.
5. Wählen Sie in der Tabelle der Geräteinstanzen die Option zum Umschalten neben der Instanz aus, zu der Sie Informationen anzeigen oder die Sie verwalten möchten, und wählen Sie dann Bearbeiten aus.



6. Um ein Instanzprofil an die Geräteinstanz anzuhängen, wählen Sie es aus der Drop-down-Liste Profil aus. Das Anhängen eines Instanzprofils kann beispielsweise hilfreich sein, wenn Sie ein bestimmtes App-Paket immer von Bereinigungsaufgaben ausschließen möchten. Weitere Informationen zur Verwendung von Instanzprofilen mit Geräten finden Sie unter. Erstellen eines Instanzprofils in AWS Device Farm
7. (Optional) Wählen Sie unter Labels (Bezeichnungen) die Option Add new (Neue hinzufügen) aus, um der Geräte-Instance eine neue Bezeichnung hinzuzufügen. Mithilfe von Bezeichnungen können Sie Geräte leichter kategorisieren und spezifische Geräte einfacher finden.
8. Wählen Sie Speichern.

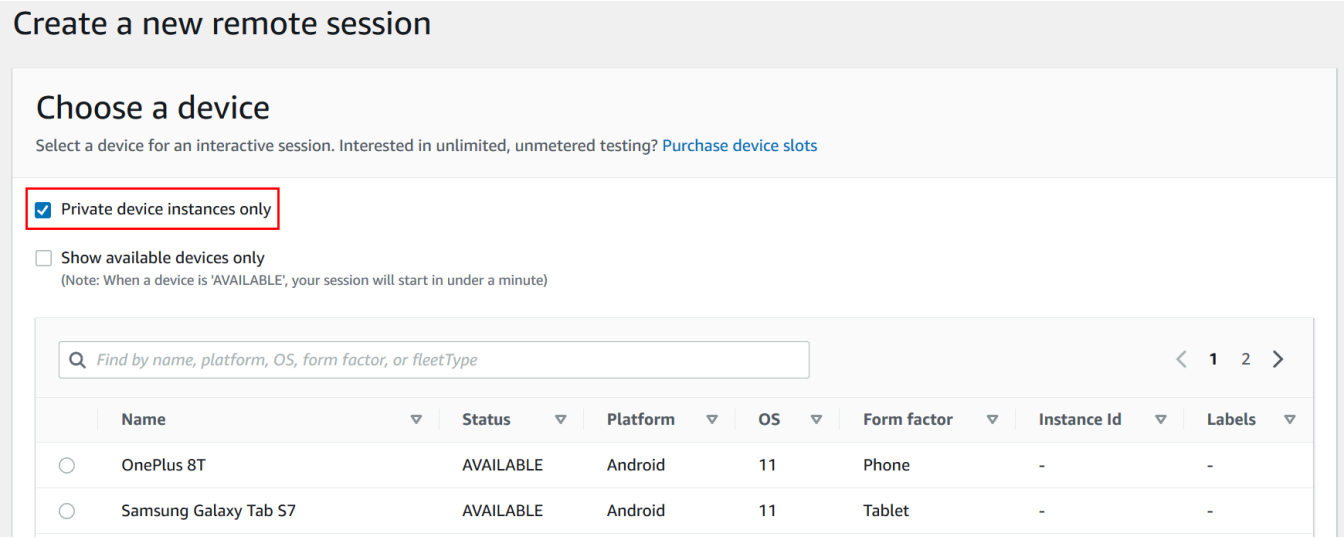
Einen Testlauf erstellen oder eine Fernzugriffssitzung in AWS Device Farm starten

In AWS Device Farm können Sie, nachdem Sie eine private Geräteflotte eingerichtet haben, Testläufe erstellen oder Fernzugriffssitzungen mit einem oder mehreren privaten Geräten in Ihrer Flotte starten. Weitere Informationen zu privaten Geräten finden Sie unterPrivate Geräte in AWS Device Farm.

Um einen Testlauf zu erstellen oder eine Fernzugriffssitzung zu starten

1. Öffnen Sie die Device Farm Farm-Konsole unter https://console.aws.amazon.com/devicefarm/.
2. Wählen Sie im Navigationsbereich Device Farm die Option Mobile Device Testing und dann Projects aus.
3. Wählen Sie ein vorhandenes Projekt aus der Liste aus oder erstellen Sie ein neues. Um ein neues Projekt zu erstellen, wählen Sie Neues Projekt aus, geben Sie einen Namen für das Projekt ein und klicken Sie dann auf Absenden.
4. Führen Sie eine der folgenden Aktionen aus:

   - Wählen Sie zum Erstellen eines Testlaufs Automated tests (Automatisierte Tests) und anschließend Create a new run (Einen neuen Testlauf erstellen) aus. Der Assistent führt Sie durch die Schritte zum Erstellen des Testlaufs. Im Schritt Geräte auswählen können Sie einen vorhandenen Gerätepool bearbeiten oder einen neuen Gerätepool erstellen, der nur die privaten Geräte enthält, die das Device Farm Farm-Team eingerichtet und mit Ihrem AWS Konto verknüpft hat. Weitere Informationen finden Sie unter the section called “Erstellen Sie einen privaten Gerätepool”.
   - Wählen Sie für eine Remote-Zugriffssitzung Remote access (Remote-Zugriff) und dann Start a new Session (Starten einer neuen Sitzung) aus. Wählen Sie auf der Seite Gerät auswählen die Option Nur private Geräteinstanzen aus, um die Liste auf die privaten Geräte zu beschränken, die das Device Farm Farm-Team eingerichtet und mit Ihrem AWS Konto verknüpft hat. Wählen Sie anschließend die Geräte aus, auf die Sie während der Remote-Zugriffssitzung zugreifen möchten, und klicken Sie auf Confirm and start session (Bestätigen und Sitzung starten).



Auswahl privater Geräte in einem Gerätepool in AWS Device Farm

Um private Geräte in Ihrem Testlauf zu verwenden, können Sie einen Gerätepool erstellen, der Ihre privaten Geräte auswählt. Gerätepools ermöglichen es Ihnen, private Geräte hauptsächlich anhand von drei Arten von Gerätepoolregeln auszuwählen:

1. Regeln, die auf dem Geräte-ARN basieren
2. Regeln, die auf dem Geräteinstanzlabel basieren
3. Regeln, die auf dem ARN der Geräteinstanz basieren

In den folgenden Abschnitten werden die einzelnen Regeltypen und ihre Anwendungsfälle ausführlich beschrieben. Sie können die Device Farm Farm-Konsole, die AWS Befehlszeilenschnittstelle (AWS CLI) oder die Device Farm Farm-API verwenden, um mithilfe dieser Regeln einen Gerätepool mit privaten Geräten zu erstellen oder zu ändern.

Themen

Geräte-ARN

Ein Geräte-ARN ist eine Kennung, die eher für einen Gerätetyp als für eine bestimmte physische Geräteinstanz steht. Ein Gerätetyp wird durch die folgenden Attribute definiert:

- Die Flotten-ID des Geräts
- Das Gerät ist OEM
- Die Modellnummer des Geräts
- Die Betriebssystemversion des Geräts
- Der Status des Geräts, der angibt, ob es gerootet ist oder nicht

Viele physische Geräteinstanzen können durch einen einzigen Gerätetyp dargestellt werden, wobei jede Instanz dieses Typs dieselben Werte für diese Attribute hat. Wenn Sie beispielsweise drei *`Apple iPhone 13`* Geräte mit iOS-Version *`16.1.0`* in Ihrer privaten Flotte hätten, würde jedes Gerät denselben Geräte-ARN teilen. Wenn Geräte mit denselben Attributen zu Ihrer Flotte hinzugefügt oder daraus entfernt würden, würde der Geräte-ARN weiterhin die verfügbaren Geräte darstellen, die Sie in Ihrer Flotte für diesen Gerätetyp hatten.

Der Geräte-ARN ist die robusteste Methode zur Auswahl privater Geräte für einen Gerätepool, da er es dem Gerätepool ermöglicht, weiterhin Geräte auszuwählen, unabhängig von den spezifischen Geräteinstanzen, die Sie zu einem bestimmten Zeitpunkt bereitgestellt haben. Bei einzelnen privaten Geräteinstanzen kann es zu Hardwarefehlern kommen, sodass Device Farm sie automatisch durch neue funktionierende Instanzen desselben Gerätetyps ersetzt. In diesen Szenarien stellt die Geräte-ARN-Regel sicher, dass Ihr Gerätepool bei einem Hardwarefehler weiterhin Geräte auswählen kann.

Wenn Sie eine Geräte-ARN-Regel für private Geräte in Ihrem Gerätepool verwenden und einen Testlauf mit diesem Pool planen, überprüft Device Farm automatisch, welche privaten Geräteinstanzen durch diesen Geräte-ARN repräsentiert werden. Von den derzeit verfügbaren Instanzen wird eine davon für die Ausführung Ihres Tests zugewiesen. Wenn derzeit keine Instanzen verfügbar sind, wartet Device Farm, bis die erste verfügbare Instanz dieses Geräte-ARN verfügbar ist, und weist sie zur Ausführung Ihres Tests zu.

Labels der Geräteinstanzen

Ein Geräteinstanzlabel ist eine Textkennung, die Sie als Metadaten für eine Geräteinstanz anhängen können. Sie können jeder Geräteinstanz mehrere Labels und mehreren Geräteinstanzen dasselbe

Label zuordnen. Weitere Informationen zum Hinzufügen, Ändern oder Entfernen von Gerätelabels zu Geräteinstanzen finden Sie unter Private Geräte verwalten.

Das Geräteinstanz-Label kann eine zuverlässige Methode zur Auswahl privater Geräte für einen Gerätepool sein, denn wenn Sie mehrere Geräteinstanzen mit derselben Bezeichnung haben, ermöglicht es dem Gerätepool, aus einer beliebigen von ihnen für Ihren Test auszuwählen. Wenn der Geräte-ARN keine gute Regel für Ihren Anwendungsfall ist (wenn Sie beispielsweise aus Geräten mehrerer Gerätetypen auswählen möchten oder wenn Sie aus einer Teilmenge aller Geräte eines Gerätetyps auswählen möchten), können Sie mithilfe von Geräteinstanzbezeichnungen detaillierter aus mehreren Geräten für Ihren Gerätepool auswählen. Bei einzelnen privaten Geräteinstanzen kann es zu Hardwarefehlern kommen, sodass Device Farm sie automatisch durch neue funktionierende Instanzen desselben Gerätetyps ersetzt. In diesen Szenarien behält die Ersatzgeräteinstanz keine Metadaten zur Instanzbezeichnung des ersetzten Geräts bei. Wenn Sie also dasselbe Geräteinstanzlabel auf mehrere Geräteinstanzen anwenden, stellt die Regel zur Geräteinstanzkennzeichnung sicher, dass Ihr Gerätepool bei einem Hardwarefehler weiterhin Geräteinstanzen auswählen kann.

Wenn Sie eine Geräteinstanz-Label-Regel für private Geräte in Ihrem Gerätepool verwenden und einen Testlauf mit diesem Pool planen, überprüft Device Farm automatisch, welche privaten Geräteinstanzen durch dieses Geräteinstanz-Label repräsentiert werden, und wählt aus diesen Instanzen nach dem Zufallsprinzip eine aus, die für die Ausführung Ihres Tests verfügbar ist. Wenn keine verfügbar sind, wählt Device Farm nach dem Zufallsprinzip eine Geräteinstanz mit der Bezeichnung Geräteinstanz aus, um Ihren Test auszuführen, und stellt den Test in die Warteschlange, um ihn auf dem Gerät auszuführen, sobald er verfügbar ist.

Instance-ARN

Eine Geräteinstanz ARN ist eine Kennung, die eine physische Bare-Metal-Geräteinstanz darstellt, die in einer privaten Flotte eingesetzt wird. Wenn Sie beispielsweise `15.0.0` in Ihrer privaten Flotte drei `iPhone 13` Geräte mit Betriebssystem hätten und jedes Gerät denselben Geräte-ARN verwenden würde, hätte jedes Gerät auch seinen eigenen Instanz-ARN, der nur diese Instanz repräsentiert.

Die Geräteinstanz ARN ist die am wenigsten robuste Methode zur Auswahl privater Geräte für einen Gerätepool und wird nur empfohlen, wenn die Geräte ARNs - und Geräteinstanzbezeichnungen nicht zu Ihrem Anwendungsfall passen. Geräteinstanzen ARNs werden häufig als Regeln für Gerätepools verwendet, wenn eine bestimmte Geräteinstanz als Voraussetzung für Ihren Test auf einzigartige und spezifische Weise konfiguriert ist und wenn diese Konfiguration bekannt und verifiziert sein muss, bevor der Test darauf ausgeführt wird. Bei einzelnen privaten Geräteinstanzen kann es

zu Hardwarefehlern kommen, sodass Device Farm sie automatisch durch neue funktionierende Instanzen desselben Gerätetyps ersetzt. In diesen Szenarien hat die Ersatzgeräteinstanz einen anderen Geräteinstanz-ARN als das ersetzte Gerät. Wenn Sie sich also ARNs für Ihren Gerätepool auf die Geräteinstanz verlassen, müssen Sie die Regeldefinition Ihres Gerätepools manuell von der Verwendung des alten ARN auf die Verwendung des neuen ARN ändern. Wenn Sie das Gerät für den Test manuell vorkonfigurieren müssen, kann dies ein effektiver Arbeitsablauf sein (im Vergleich zum Gerät ARNs). Für Tests in großem Maßstab wird empfohlen, diese Anwendungsfälle so anzupassen, dass sie mit Geräteinstanzbezeichnungen funktionieren, und wenn möglich, mehrere Geräteinstanzen für Tests vorkonfigurieren zu lassen.

Wenn Sie eine ARN-Regel für Geräteinstanzen für private Geräte in Ihrem Gerätepool verwenden und einen Testlauf mit diesem Pool planen, weist Device Farm diesen Test automatisch dieser Geräteinstanz zu. Wenn diese Geräteinstanz nicht verfügbar ist, stellt Device Farm den Test auf dem Gerät in die Warteschlange, sobald er verfügbar ist.

Erstellen eines privaten Gerätepools mit privaten Geräten (Konsole)

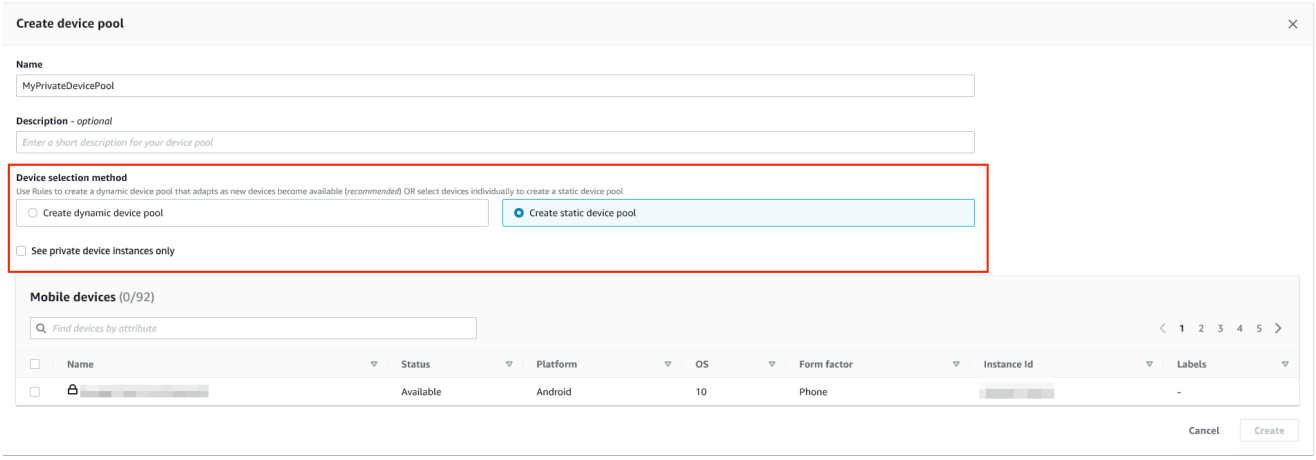
Wenn Sie einen Testlauf erstellen, können Sie einen Gerätepool für den Testlauf erstellen, und überprüfen, dass der Pool nur Ihre privaten Geräte umfasst.

> **Note**
>
> Wenn Sie einen Gerätepool mit privaten Geräten in der Konsole erstellen, können Sie nur eine der drei verfügbaren Regeln für die Auswahl privater Geräte verwenden. Wenn Sie einen Gerätepool erstellen möchten, der mehrere Arten von Regeln für private Geräte enthält (z. B. Gerätepools, die Regeln für Geräte ARNs und Geräteinstanzen enthalten ARNs), müssen Sie den Pool über die CLI oder API erstellen.

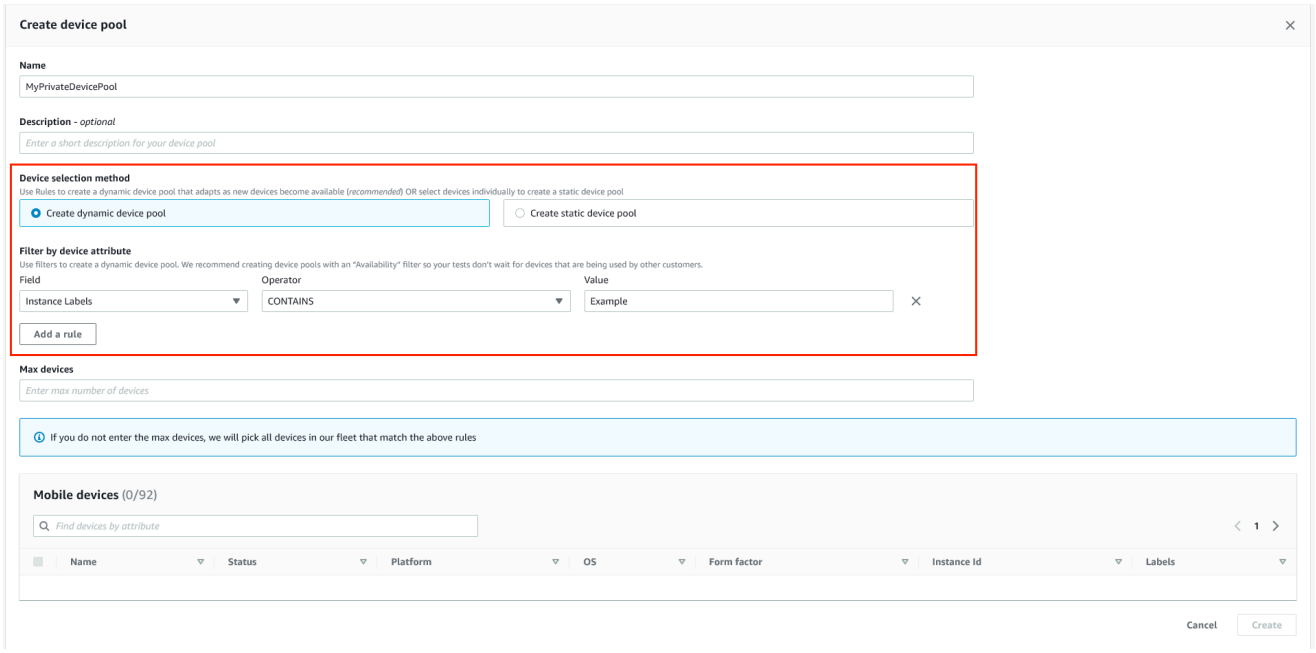
1. Öffnen Sie die Device Farm Farm-Konsole unter https://console.aws.amazon.com/devicefarm/.
2. Wählen Sie im Navigationsbereich Device Farm die Option Mobile Device Testing und dann Projects aus.
3. Wählen Sie ein vorhandenes Projekt aus der Liste aus oder erstellen Sie ein neues. Um ein neues Projekt zu erstellen, wählen Sie Neues Projekt aus, geben Sie einen Namen für das Projekt ein und klicken Sie dann auf Absenden.
4. Wählen Sie Projekteinstellungen und navigieren Sie dann zur Registerkarte Gerätepools.

5. Wählen Sie Gerätepool erstellen und geben Sie einen Namen und eine optionale Beschreibung für Ihren Gerätepool ein.

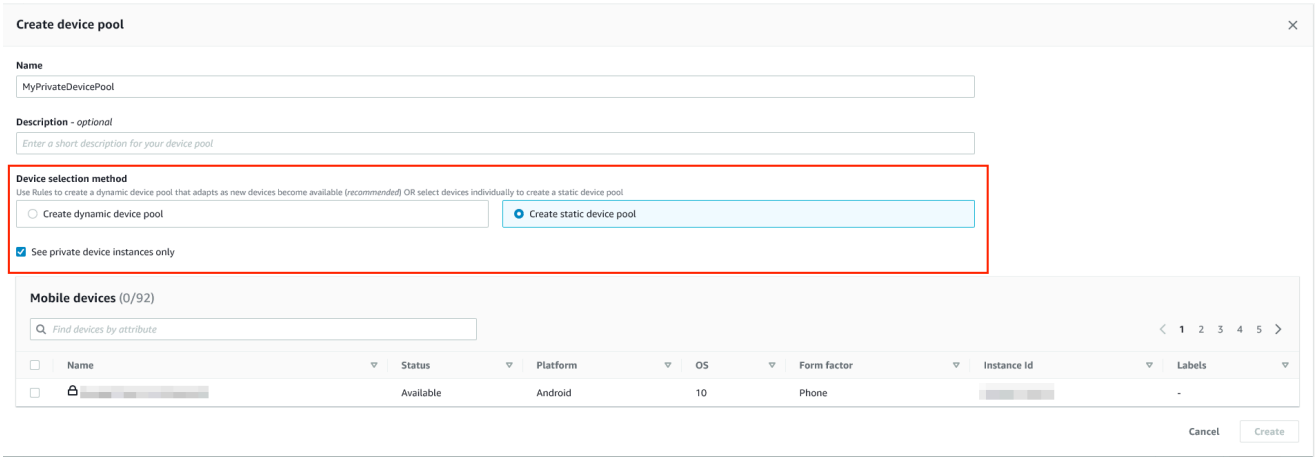
   a. Um Geräte-ARN-Regeln für Ihren Gerätepool zu verwenden, wählen Sie Statischen Gerätepool erstellen und wählen Sie dann die spezifischen Gerätetypen aus der Liste aus, die Sie im Gerätepool verwenden möchten. Wählen Sie Private Geräteinstanzen nicht nur aus, weil diese Option bewirkt, dass der Gerätepool mit ARN-Regeln für Geräteinstanzen (anstelle von Geräte-ARN-Regeln) erstellt wird.



   b. Um Labelregeln für Geräteinstanzen für Ihren Gerätepool zu verwenden, wählen Sie Dynamischen Gerätepool erstellen. Wählen Sie dann für jedes Label, das Sie im Gerätepool verwenden möchten, die Option Regel hinzufügen aus. Wählen Sie für jede Regel Instance Labels als`Field`, wählen Sie Contains als aus und geben Sie das gewünschte Geräteinstanzlabel als an`Value`. `Operator`



c. Um ARN-Regeln für Geräteinstanzen für Ihren Gerätepool zu verwenden, wählen Sie Statischen Gerätepool erstellen und anschließend Nur private Geräteinstanzen aus, um die Geräteliste auf die privaten Geräteinstanzen zu beschränken, die Device Farm Ihrem AWS Konto zugeordnet hat.



6. Wählen Sie Erstellen aus.

Einen privaten Gerätepool mit privaten Geräten erstellen (AWS CLI)

- Führen Sie den Befehl create-device-pool aus.

Informationen zur Verwendung von Device Farm mit dem AWS CLI finden Sie unterAWS CLI Referenz.

Erstellen eines privaten Gerätepools mit privaten Geräten (API)

- Rufen Sie die `CreateDevicePool`-API auf.

Hinweise zur Verwendung der Device Farm API finden Sie unterDevice Farm automatisieren.

Überspringen der erneuten Signierung von Apps auf privaten Geräten in AWS Device Farm

Das Signieren von Apps ist ein Prozess, bei dem ein App-Paket (z. B. APK, IPA) mit einem privaten Schlüssel digital signiert wird, bevor es auf einem Gerät installiert oder in einem App Store wie dem Google Play Store oder dem Apple App Store veröffentlicht werden kann. Um das Testen zu

optimieren, indem die Anzahl der benötigten Signaturen und Profile reduziert und die Datensicherheit auf Remote-Geräten erhöht wird, signiert AWS Device Farm Ihre App erneut, nachdem sie in den Service hochgeladen wurde.

Sobald Sie Ihre App auf AWS Device Farm hochgeladen haben, generiert der Service mithilfe seiner eigenen Signaturzertifikate und Bereitstellungsprofile eine neue Signatur für die App. Dieser Prozess ersetzt die ursprüngliche App-Signatur durch die Signatur von AWS Device Farm. Die neu signierte App wird dann auf den von AWS Device Farm bereitgestellten Testgeräten installiert. Die neue Signatur ermöglicht die Installation und Ausführung der App auf diesen Geräten, ohne dass die ursprünglichen Entwicklerzertifikate erforderlich sind.

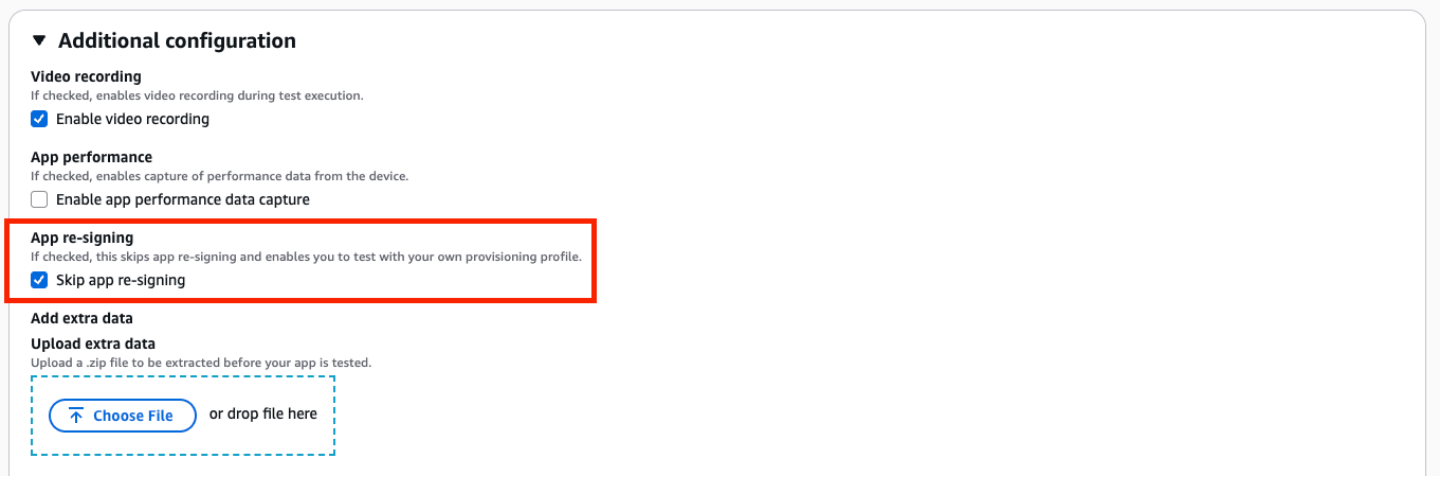
Auf iOS ersetzen wir das eingebettete Bereitstellungsprofil durch ein Platzhalterprofil und signieren die App erneut. Wenn Sie es angeben, fügen wir dem Anwendungspaket vor der Installation zusätzliche Daten hinzu, sodass die Daten in der Sandbox Ihrer App vorhanden sind. Durch erneutes Signieren der iOS-App werden alle Berechtigungen entfernt.

Auf Android signieren wir die App erneut. Dadurch können Funktionen beeinträchtigt werden, die von der App-Signatur abhängen, wie z. B. die Google Maps Android API. Es kann auch die Erkennung von Piraterie und Manipulation auslösen, die in Produkten wie zum Beispiel verfügbar sind. DexGuard Bei integrierten Tests können wir das Manifest so ändern, dass es die erforderlichen Berechtigungen für die Erfassung und Speicherung von Screenshots enthält.

Wenn Sie private Geräte verwenden, können Sie den Schritt überspringen, bei dem AWS Device Farm Ihre App erneut signiert. Dies unterscheidet sich von öffentlichen Geräten, bei denen Device Farm Ihre App auf den Android- und iOS-Plattformen immer neu signiert.

Sie können die erneute App-Signatur überspringen, wenn Sie eine Remote-Zugriffssitzung oder einen Testlauf erstellen . Dies kann hilfreich sein, wenn Ihre App über Funktionen verfügt, die nicht mehr funktionieren, wenn Device Farm Ihre App erneut signiert. Beispielsweise funktionieren Push-Benachrichtigungen möglicherweise nicht mehr, nachdem eine erneute Signatur durchgeführt wurde. Weitere Informationen zu den Änderungen, die Device Farm beim Testen Ihrer App vornimmt, finden Sie auf der Seite AWS Device Farm FAQs oder Apps.

Um das erneute Signieren von Apps für einen Testlauf zu überspringen, wählen Sie unter Zusätzliche Konfiguration die Option App-Neusignierung überspringen aus. Diese Option ist nur für private Geräte verfügbar.



> **Note**
>
> Wenn Sie das XCTest Framework verwenden, ist die Option App erneut signieren nicht verfügbar. Weitere Informationen finden Sie unter Integrieren von Device Farm mit XCTest für iOS.

Zusätzliche Schritte zum Konfigurieren Ihrer App-Signatureinstellungen variieren, je nachdem, ob Sie private Android- oder iOS-Geräte verwenden.

Das erneute Signieren von Apps auf Android-Geräten wird übersprungen

Beim Testen Ihrer App auf einem privaten Android-Gerät, wählen Sie Skip app re-signing (Überspringen der erneuten Signatur von Apps), wenn Sie Ihren Testlauf oder Ihre Remote-Zugriffssitzung erstellen. Weitere Konfigurationsarbeiten sind nicht erforderlich.

Das erneute Signieren von Apps auf iOS-Geräten wird übersprungen

Apple erfordert eine Signatur zum Testen der App, bevor Sie sie auf ein Gerät laden können. Für iOS-Geräte haben Sie zwei Möglichkeiten, Ihre App zu signieren.

- Wenn Sie ein internes Entwickler-Profil (Enterprise) verwenden, können Sie diesen Schritt überspringen und mit dem nächsten Abschnitt, the section called “Erstellen Sie eine Fernzugriffssitzung, um Ihrer App zu vertrauen”, fortfahren.
- Wenn Sie ein Ad-hoc-Entwicklungsprofil für iOS-Apps verwenden, müssen Sie zunächst das Gerät bei Ihrem Apple-Entwicklerkonto registrieren und anschließend Ihr Bereitstellungsprofil

mit dem privaten Gerät aktualisieren. Anschließend müssen Sie Ihre App mit dem aktualisierten Bereitstellungsprofil erneut signieren. Anschließend können Sie Ihre neu signierte App in Device Farm ausführen.

So registrieren Sie ein Gerät mit einem Ad-hoc-Entwicklungs-Bereitstellungsprofil für iOS-Apps

1. Melden Sie sich bei Ihrem Apple-Entwicklerkonto an.
2. Navigieren Sie in der Konsole zum Bereich Zertifikate IDs, und Profile.
3. Gehen Sie zu Devices (Geräte).
4. Registrieren Sie das Geräts bei Ihrem Apple-Entwicklerkonto. Um den Namen und die UDID des Geräts abzurufen, verwenden Sie den `ListDeviceInstances` Betrieb der Device Farm API.
5. Gehen Sie zu Ihrem Bereitstellungsprofil und wählen Sie Edit (Bearbeiten) aus.
6. Wählen Sie das Gerät aus der Liste aus.
7. Rufen Sie in Xcode Ihr aktualisiertes Bereitstellungsprofil ab und signieren Sie die App erneut.

Weitere Konfigurationsarbeiten sind nicht erforderlich. Sie können jetzt eine Remote-Zugriffssitzung oder einen Testlauf erstellen und dann Skip app re-signing (Überspringen der erneuten Signatur von Apps) auswählen.

Eine Fernzugriffssitzung erstellen, um Ihrer iOS-App zu vertrauen

Wenn Sie ein internes Entwickler-Bereitstellungsprofil (Enterprise) verwenden, müssen Sie ein einmaliges Verfahren ausführen, um dem internen App-Entwicklerzertifikat auf jedem Ihrer Geräte zu vertrauen.

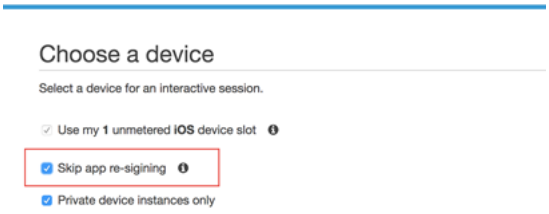
Dazu müssen Sie eine Platzhalter-App installieren, die mit demselben Zertifikat signiert ist wie die App, die Sie testen möchten. Wenn das Gerät dem Konfigurationsprofil oder dem Entwickler der Unternehmens-App vertraut, gelten alle Apps dieses Entwicklers auf dem privaten Gerät als vertrauenswürdig, bis Sie sie löschen. Wenn Sie also neue Versionen der App installieren, die Sie testen möchten, müssen Sie dem App-Entwickler nicht jedes Mal erneut vertrauen. Dies ist besonders nützlich, wenn Sie Testautomatisierungen ausführen und nicht bei jedem Testen Ihrer App eine Remote-Zugriffssitzung erstellen wollen.

Ein gängiges Verfahren, das viele Kunden verwenden, besteht darin, die Device Farm Farm-Beispiel-App für iOS erneut zu signieren und sie dann als Platzhalter-App auf ihrem Gerät zu installieren.

Bevor Sie Ihre Fernzugriffssitzung starten, folgen Sie den Schritten unter Erstellen eines Instanzprofils in AWS Device Farm So erstellen oder ändern Sie ein Instanzprofil in Device Farm. Fügen Sie im Instanzprofil die Bundle-ID der Platzhalter-App zur Einstellung Pakete von der Bereinigung ausschließen hinzu. Hängen Sie dann das Instanzprofil an die private Geräteinstanz an, um sicherzustellen, dass Device Farm diese App nicht vom Gerät entfernt, bevor ein neuer Testlauf gestartet wird. Auf diese Weise wird sichergestellt, dass Ihr Entwicklerzertifikat vertrauenswürdig bleibt.

Sie können die Platzhalter-App mithilfe einer Fernzugriffssitzung auf das Gerät hochladen, sodass Sie die App starten und dem Entwickler vertrauen können.

1. Befolgen Sie die Anweisungen unter Erstellen einer Sitzung, um eine Remote-Zugriffssitzung zu erstellen. Verwenden Sie dazu das soeben erstellte Instance-Profil des privaten Geräts. Wenn Sie Ihre Sitzung erstellen, achten Sie darauf, Skip app re-signing (Überspringen der erneuten Signatur von Apps) auszuwählen.



> **Important**
>
> Filtern Sie die Liste der Geräte nach privaten Geräten, indem Sie Private device instances only (Nur private Geräte-Instances) auswählen. So stellen Sie sicher, dass Sie ein privates Gerät mit dem richtigen Instance-Profil verwenden.

   Stellen Sie sicher, dass Sie auch die Platzhalter-App oder die App, die Sie testen möchten, zur Einstellung Pakete von der Bereinigung ausschließen für das Instanzprofil hinzufügen, das mit dieser Instanz verknüpft ist.
2. Wenn Ihre Remotesitzung gestartet wird, wählen Sie Datei auswählen, um eine Anwendung zu installieren, die Ihr internes Bereitstellungsprofil verwendet.
3. Starten Sie die App, die Sie gerade hochgeladen haben.
4. Vergewissern Sie sich, dass ein iOS-Dialogfeld angezeigt wird, das darauf hinweist, dass dem Entwickler der Unternehmens-App nicht vertraut wird.

5. Wenn auf dem iOS-Gerät iOS Version 18 oder höher installiert ist, öffnen Sie ein Supportticket beim AWS Device Farm Farm-Team, damit unser Team der App für Sie vertraut, da für diese Geräte der App manuell vertraut werden muss. Andernfalls, wenn die iOS-Version 17 oder niedriger ist, können Sie in die Einstellungen-App gehen und unter Allgemeine Einstellungen der App im Menü VPN und Profile selbst vertrauen.

Alle Apps von diesem "Configuration Profile" (Konfigurationsprofil)- oder Enterprise App-Entwickler sind jetzt auf diesem privaten Gerät vertrauenswürdig, bis Sie sie löschen.

Amazon VPC AWS regionsübergreifend in AWS Device Farm

Device Farm Farm-Dienste befinden sich nur in der Region USA West (Oregon`us-west-2`) (). Sie können Amazon Virtual Private Cloud (Amazon VPC) verwenden, um mithilfe von Device Farm einen Service in Ihrer Amazon Virtual Private Cloud in einer anderen AWS Region zu erreichen. Wenn sich Device Farm und Ihr Dienst in derselben Region befinden, finden Sie weitere Informationen unterVerwendung von Amazon VPC-Endpunktservices mit Device Farm — Legacy (nicht empfohlen).

Es gibt zwei Möglichkeiten, auf Ihre privaten Dienste zuzugreifen, die sich in einer anderen Region befinden. Wenn sich Dienste in einer anderen Region befinden, in der dies nicht der Fall ist`us-west-2`, können Sie VPC Peering verwenden, um die VPC dieser Region mit einer anderen VPC zu verbinden, die mit Device Farm in verbunden ist. `us-west-2` Wenn Sie jedoch Dienste in mehreren Regionen haben, können Sie mit einem Transit Gateway mit einer einfacheren Netzwerkkonfiguration auf diese Dienste zugreifen.

Weitere Informationen finden Sie unter VPC-Peering-Szenarien im Amazon VPC Peering Guide.

Überblick über das VPC-Peering für VPCs verschiedene Regionen in AWS Device Farm

Sie können zwei beliebige CIDR-Blöcke VPCs in verschiedenen Regionen miteinander verbinden, sofern sie unterschiedliche, sich nicht überlappende CIDR-Blöcke haben. Dadurch wird sichergestellt, dass alle privaten IP-Adressen eindeutig sind, und alle Ressourcen in der können sich gegenseitig adressieren, ohne dass irgendeine Form von Network Address Translation (NAT) erforderlich ist. VPCs Weitere Informationen zur CIDR-Notation finden Sie unter RFC 4632.

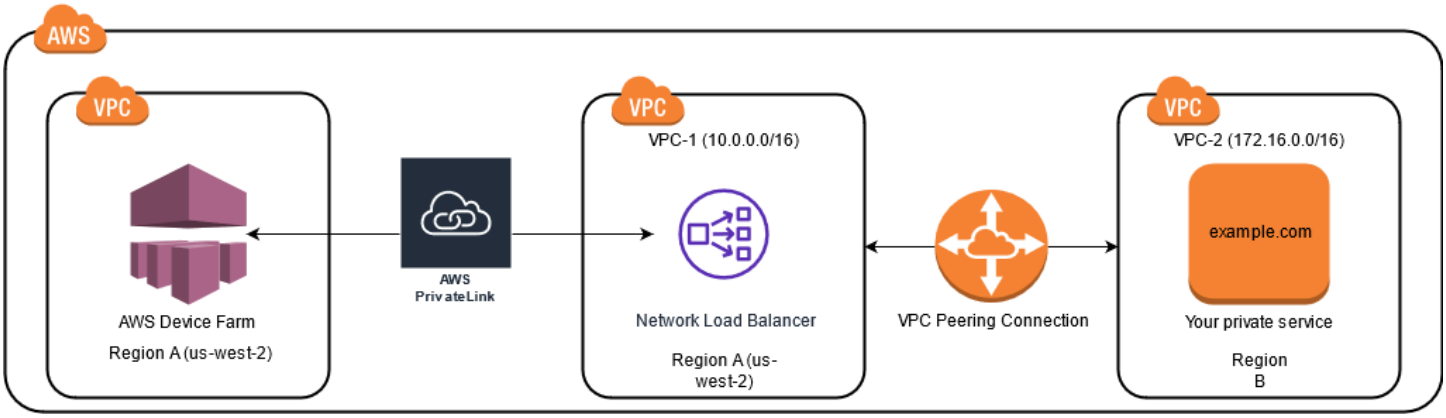
Dieses Thema enthält ein regionsübergreifendes Beispielszenario, in dem sich die Device Farm (als VPC-1 bezeichnet) in der Region USA West (Oregon) (`us-west-2`) befindet. Die zweite VPC in diesem Beispiel (als VPC-2 bezeichnet) befindet sich in einer anderen Region.

Beispiel für eine regionsübergreifende Device Farm VPC

VPC-Komponente	VPC-1	VPC-2
CIDR	10.0.0.0/16	172.16.0.0/16

> **Important**
>
> Das Herstellen einer Peering-Verbindung zwischen zwei VPCs kann den Sicherheitsstatus von ändern. VPCs Darüber hinaus kann das Hinzufügen neuer Einträge zu ihren Routing-Tabellen den Sicherheitsstatus der Ressourcen innerhalb von ändern. VPCs Es liegt in Ihrer Verantwortung, diese Konfigurationen so zu implementieren, dass sie den Sicherheitsanforderungen Ihres Unternehmens entsprechen. Weitere Informationen finden Sie im Modell der geteilten Verantwortung.

Das folgende Diagramm zeigt die Komponenten dieses Beispiels und die Interaktionen zwischen diesen Komponenten.



Themen

- Voraussetzungen für die Verwendung von Amazon VPC in AWS Device Farm
- Schritt 1: Einrichten einer Peering-Verbindung zwischen VPC-1 und VPC-2
- Schritt 2: Aktualisierung der Routentabellen in VPC-1 und VPC-2
- Schritt 3: Eine Zielgruppe erstellen
- Schritt 4: Einen Network Load Balancer erstellen
- Schritt 5: Einen VPC-Endpunktdienst erstellen, um Ihre VPC mit der Device Farm zu verbinden

- Schritt 6: Erstellen Sie eine VPC-Endpunktkonfiguration zwischen Ihrer VPC und der Device Farm
- Schritt 7: Erstellen eines Testlaufs zur Verwendung der VPC-Endpunktkonfiguration
- Aufbau eines skalierbaren Netzwerks mit Transit Gateway

Voraussetzungen für die Verwendung von Amazon VPC in AWS Device Farm

Für dieses Beispiel müssen die folgenden Voraussetzungen erfüllt sein:

- Zwei VPCs , die mit Subnetzen konfiguriert sind, die sich nicht überlappende CIDR-Blöcke enthalten.
- VPC-1 muss sich in der `us-west-2` Region befinden und Subnetze für Availability Zones`us-west-2a`, `us-west-2b` und enthalten. `us-west-2c`

Weitere Informationen zum Erstellen VPCs und Konfigurieren von Subnetzen finden Sie unter Arbeiten mit VPCs und Subnetzen im Amazon VPC Peering Guide.

Schritt 1: Einrichten einer Peering-Verbindung zwischen VPC-1 und VPC-2

Stellen Sie eine Peering-Verbindung zwischen den beiden her, die sich nicht überlappende VPCs CIDR-Blöcke enthält. Informationen dazu finden Sie unter Erstellen und Akzeptieren von VPC-Peering-Verbindungen im Amazon VPC Peering Guide. Anhand des regionsübergreifenden Szenarios dieses Themas und des Amazon VPC Peering Guide wird die folgende Beispielkonfiguration für eine Peering-Verbindung erstellt:

Name

```
Device-Farm-Peering-Connection-1
```

VPC-ID (Anforderer)

```
vpc-0987654321gfedcba (VPC-2)
```

Account

```
My account
```

Region

```
US West (Oregon) (us-west-2)
```

VPC-ID (Akzeptierer)

```
vpc-1234567890abcdefg (VPC-1)
```

> **Note**
>
> Stellen Sie sicher, dass Sie Ihre VPC-Peering-Verbindungskontingente beachten, wenn Sie neue Peering-Verbindungen einrichten. Weitere Informationen finden Sie unter Amazon VPC-Kontingente im Amazon VPC Peering Guide.

Schritt 2: Aktualisierung der Routentabellen in VPC-1 und VPC-2

Nachdem Sie eine Peering-Verbindung eingerichtet haben, müssen Sie eine Zielroute zwischen den beiden einrichten, um Daten zwischen ihnen VPCs zu übertragen. Um diese Route einzurichten, können Sie die Routentabelle von VPC-1 manuell aktualisieren, sodass sie auf das Subnetz von VPC-2 verweist und umgekehrt. Lesen Sie dazu den Abschnitt Aktualisieren Ihrer Routentabellen für eine VPC-Peering-Verbindung im Amazon VPC Peering Guide. Mithilfe des regionsübergreifenden Szenarios dieses Themas und des Amazon VPC Peering Guide wird die folgende Beispielkonfiguration für eine Routing-Tabelle erstellt:

Beispiel für eine VPC-Routentabelle für eine Device Farm

VPC-Komponente	VPC-1	VPC-2
Routing-Tabellen-ID	rtb-1234567890abcdefg	rtb-0987654321gfedcba
Lokaler Adressbereich	10.0.0.0/16	172.16.0.0/16
Zieladressbereich	172.16.0.0/16	10.0.0.0/16

Schritt 3: Eine Zielgruppe erstellen

Nachdem Sie Ihre Zielrouten eingerichtet haben, können Sie in VPC-1 einen Network Load Balancer konfigurieren, um Anfragen an VPC-2 weiterzuleiten.

Der Network Load Balancer muss zunächst eine Zielgruppe enthalten, die die IP-Adressen enthält, an die Anfragen gesendet werden.

Um eine Zielgruppe zu erstellen

1. Identifizieren Sie die IP-Adressen des Dienstes, auf den Sie in VPC-2 abzielen möchten.
   - Diese IP-Adressen müssen Mitglieder des Subnetzes sein, das für die Peering-Verbindung verwendet wird.
   - Die Ziel-IP-Adressen müssen statisch und unveränderlich sein. Wenn Ihr Service über dynamische IP-Adressen verfügt, sollten Sie erwägen, eine statische Ressource (z. B. einen Network Load Balancer) als Ziel zu verwenden und diese statische Ressource Anfragen an Ihr wahres Ziel weiterleiten zu lassen.

   > **Note**
   > - Wenn Sie eine oder mehrere eigenständige Amazon Elastic Compute Cloud (Amazon EC2) -Instances als Ziel haben, öffnen Sie die Amazon EC2-Konsole unter https://console.aws.amazon.com/ec2/und wählen Sie dann Instances aus.
   > - Wenn Sie auf eine Amazon EC2 Auto Scaling Scaling-Gruppe von Amazon EC2-Instances abzielen, müssen Sie die Amazon EC2 Auto Scaling-Gruppe einem Network Load Balancer zuordnen. Weitere Informationen finden Sie unter Anhängen eines Load Balancers an Ihre Auto-Scaling-Gruppe im Benutzerhandbuch für Amazon EC2 Auto Scaling.
   >
   > Anschließend können Sie die Amazon EC2 EC2-Konsole unter öffnen und https://console.aws.amazon.com/ec2/dann Netzwerkschnittstellen wählen. Von dort aus können Sie die IP-Adressen für jede Netzwerkschnittstelle des Network Load Balancer in jeder Availability Zone anzeigen.

2. Erstellen Sie eine Zielgruppe in VPC-1. Informationen dazu finden Sie unter Erstellen einer Zielgruppe für Ihren Network Load Balancer im Benutzerhandbuch für Network Load Balancer.

   Zielgruppen für Dienste in einer anderen VPC benötigen die folgende Konfiguration:

   - Wählen Sie für Wählen Sie einen Zieltyp die Option IP-Adressen aus.
   - Wählen Sie für VPC die VPC aus, die den Load Balancer hosten soll. Für das Themenbeispiel wird dies VPC-1 sein.
   - Registrieren Sie auf der Seite Ziele registrieren ein Ziel für jede IP-Adresse in VPC-2.

     Wählen Sie für Netzwerk die Option Andere private IP-Adresse aus.

Wählen Sie für Availability Zone Ihre gewünschten Zonen in VPC-1 aus.

Wählen Sie als IPv4 Adresse die VPC-2-IP-Adresse aus.

Wählen Sie unter Ports Ihre Ports aus.

- Wählen Sie Schließen Sie die unten angeführten als ausstehend ein aus. Wenn Sie mit der Angabe von Adressen fertig sind, wählen Sie Ausstehende Ziele registrieren aus.

Unter Verwendung des regionsübergreifenden Szenarios dieses Themas und des Benutzerhandbuchs für Network Load Balancer werden die folgenden Werte in der Zielgruppenkonfiguration verwendet:

Zieltyp

```
IP addresses
```

Name der Zielgruppe

```
my-target-group
```

Protokoll/Port

```
TCP : 80
```

VPC

```
vpc-1234567890abcdefg (VPC-1)
```

Netzwerk

```
Other private IP address
```

Availability Zone

```
all
```

IPv4 address

```
172.16.100.60
```

Ports

```
80
```

Schritt 4: Einen Network Load Balancer erstellen

Erstellen Sie einen Network Load Balancer mit der in Schritt 3 beschriebenen Zielgruppe. Informationen dazu finden Sie unter Einen Network Load Balancer erstellen.

Unter Verwendung des regionsübergreifenden Szenarios dieses Themas werden die folgenden Werte in einer Beispielkonfiguration für den Network Load Balancer verwendet:

Load balancer name

```
my-nlb
```

Schema

```
Internal
```

VPC

```
vpc-1234567890abcdefg (VPC-1)
```

Zuweisung

```
us-west-2a - subnet-4i23iuufkdiufsloi
```

```
us-west-2b - subnet-7x989pkjj78nmn23j
```

```
us-west-2c - subnet-0231ndmas12bnnsds
```

Protokoll/Port

```
TCP : 80
```

Zielgruppe

```
my-target-group
```

Schritt 5: Einen VPC-Endpunktdienst erstellen, um Ihre VPC mit der Device Farm zu verbinden

Sie können den Network Load Balancer verwenden, um einen VPC-Endpunktdienst zu erstellen. Über diesen VPC-Endpunktdienst kann Device Farm ohne zusätzliche Infrastruktur, wie z. B. ein Internet-Gateway, eine NAT-Instance oder eine VPN-Verbindung, eine Verbindung zu Ihrem Dienst in VPC-2 herstellen.

Informationen dazu finden Sie unter Amazon VPC-Endpunktservice erstellen.

Schritt 6: Erstellen Sie eine VPC-Endpunktkonfiguration zwischen Ihrer VPC und der Device Farm

Jetzt können Sie eine private Verbindung zwischen Ihrer VPC und Device Farm herstellen. Sie können Device Farm verwenden, um private Dienste zu testen, ohne sie über das öffentliche Internet verfügbar zu machen. Informationen dazu finden Sie unter Erstellen einer VPC-Endpunktkonfiguration in Device Farm.

Unter Verwendung des regionsübergreifenden Szenarios dieses Themas werden die folgenden Werte in einer beispielhaften VPC-Endpunktkonfiguration verwendet:

Name

```
My VPCE Configuration
```

Name des VPCE-Dienstes

```
com.amazonaws.vpce.us-west-2.vpce-svc-1234567890abcdefg
```

DNS-Name des Dienstes

```
devicefarm.com
```

Schritt 7: Erstellen eines Testlaufs zur Verwendung der VPC-Endpunktkonfiguration

Sie können Testläufe erstellen, die die in Schritt 6 beschriebene VPC-Endpunktkonfiguration verwenden. Für weitere Informationen siehe Einen Testlauf in Device Farm erstellen oder Erstellen einer Sitzung.

Aufbau eines skalierbaren Netzwerks mit Transit Gateway

Um ein skalierbares Netzwerk mit mehr als zwei zu erstellen VPCs, können Sie Transit Gateway als Netzwerk-Transit-Hub verwenden, um Ihre Netzwerke VPCs und Ihre lokalen Netzwerke miteinander zu verbinden. Um eine VPC in derselben Region wie Device Farm für die Verwendung eines Transit Gateway zu konfigurieren, können Sie dem Leitfaden Amazon VPC Endpoint Services with Device Farm folgen, um Ressourcen in einer anderen Region anhand ihrer privaten IP-Adressen gezielt anzusprechen.

Weitere Informationen zu Transit Gateway finden Sie unter Was ist ein Transit-Gateway? im Amazon VPC Transit Gateways Guide.

Private Geräte in Device Farm beenden

`<Um ein privates Gerät nach Ablauf der ursprünglich vereinbarten Laufzeit zu kün`
Weitere Informationen zu privaten Geräten finden Sie unter. Private Geräte in AWS Device Farm

> **Important**
>
> Diese Anweisungen gelten nur für die Kündigung von Verträgen über private Geräte. Informationen zu allen anderen AWS Diensten und Fragen zur Abrechnung finden Sie in der jeweiligen Dokumentation für diese Produkte oder wenden Sie sich an den AWS Support.

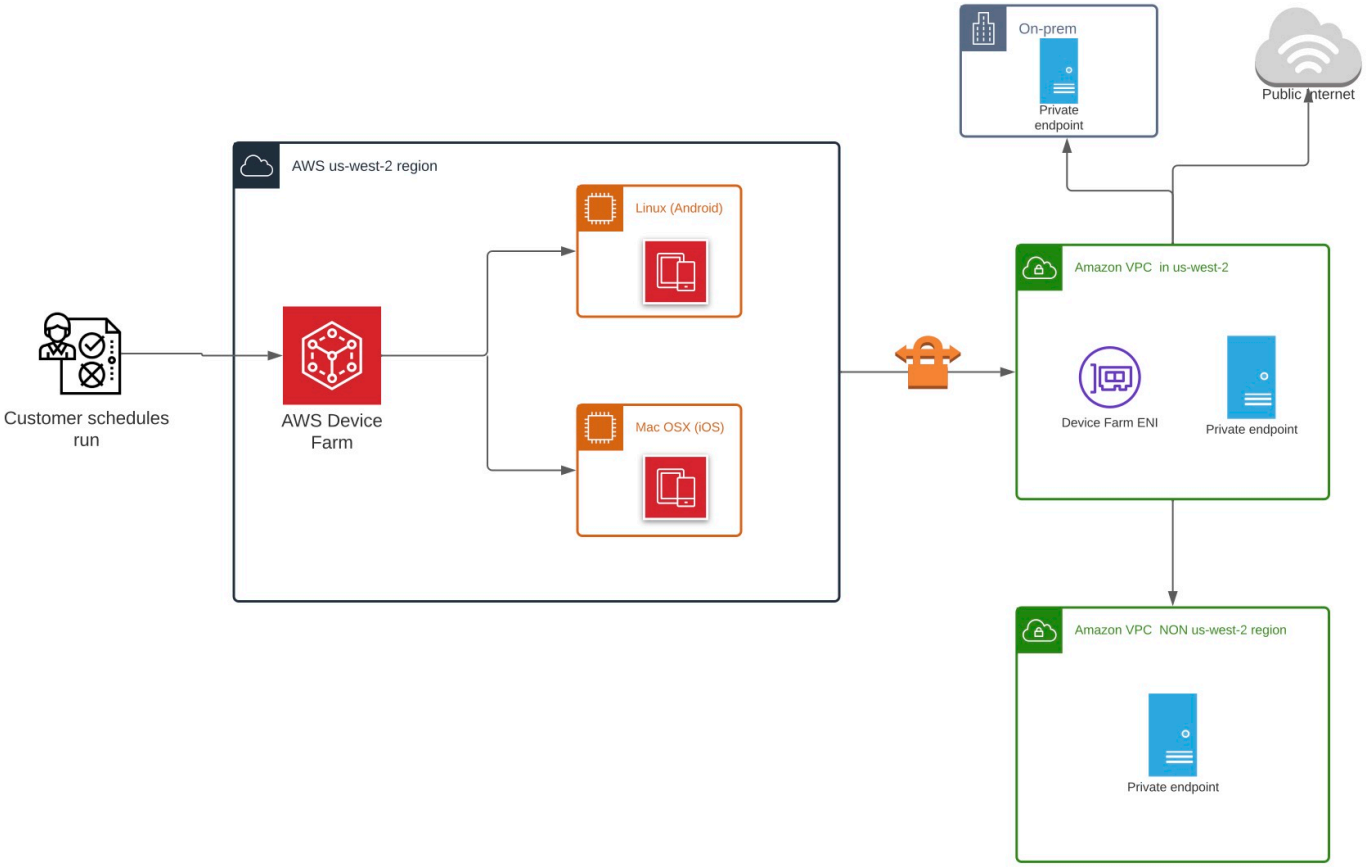
VPC-ENI in der AWS-Gerätefarm

> **Warning**
>
> Diese Funktion ist nur auf privaten Geräten verfügbar. Um die Nutzung privater Geräte auf Ihrem AWS Konto zu beantragen, kontaktieren Sie uns bitte. Wenn Sie Ihrem AWS Konto bereits private Geräte hinzugefügt haben, empfehlen wir dringend, diese Methode der VPC-Konnektivität zu verwenden.

Die VPC-ENI-Konnektivitätsfunktion von AWS Device Farm hilft Kunden dabei, eine sichere Verbindung zu ihren privaten Endpunkten herzustellen AWS, die auf einer lokalen Software oder einem anderen Cloud-Anbieter gehostet werden.

Sie können sowohl Device Farm Farm-Mobilgeräte als auch deren Host-Computer mit einer Amazon Virtual Private Cloud (Amazon VPC) -Umgebung in der `us-west-2` Region verbinden, die den Zugriff auf isolierte non-internet-facing Dienste und Anwendungen über eine elastic network interface ermöglicht. Weitere Informationen VPCs finden Sie im Amazon VPC-Benutzerhandbuch.

Wenn sich Ihr privater Endpunkt oder Ihre VPC nicht in der `us-west-2` Region befindet, können Sie sie mithilfe von Lösungen wie einem Transit Gateway oder VPC-Peering mit einer VPC in der `us-west-2` Region verknüpfen. In solchen Situationen erstellt Device Farm eine ENI in einem Subnetz, das Sie für Ihre `us-west-2` Regions-VPC bereitstellen, und Sie sind dafür verantwortlich, dass eine Verbindung zwischen der `us-west-2` Regions-VPC und der VPC in der anderen Region hergestellt werden kann.



Informationen zur automatischen Erstellung und AWS CloudFormation zum Erstellen von Peers VPCs finden Sie in den VPCPeering Vorlagen im Vorlagen-Repository unter AWS CloudFormation . GitHub

> **Note**
>
> Device Farm berechnet nichts für die Erstellung ENIs in der VPC eines Kunden in `us-west-2`. Die Kosten für regionsübergreifende oder externe VPC-Inter-VPC-Konnektivität sind in dieser Funktion nicht enthalten.

Sobald Sie den VPC-Zugriff konfiguriert haben, können die Geräte und Hostmaschinen, die Sie für Ihre Tests verwenden, keine Verbindung zu Ressourcen außerhalb der VPC (z. B. öffentlich CDNs)

herstellen, es sei denn, es gibt ein NAT-Gateway, das Sie innerhalb der VPC angeben. Weitere Informationen finden Sie unter NAT-Gateways im Benutzerhandbuch für Amazon VPC.

Themen

AWS Zugriffskontrolle und IAM

Mit AWS Device Farm können Sie AWS Identity and Access Management(IAM) Richtlinien erstellen, die den Zugriff auf die Funktionen von Device Farm gewähren oder einschränken. Um die VPC-Konnektivitätsfunktion mit AWS Device Farm zu verwenden, ist die folgende IAM-Richtlinie für das Benutzerkonto oder die Rolle erforderlich, die Sie für den Zugriff auf AWS Device Farm verwenden:

JSON

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

Um ein Device Farm Farm-Projekt mit einer VPC-Konfiguration zu erstellen oder zu aktualisieren, muss Ihre IAM-Richtlinie es Ihnen ermöglichen, die folgenden Aktionen für die in der VPC-Konfiguration aufgeführten Ressourcen aufzurufen:

```
"ec2:DescribeVpcs"
"ec2:DescribeSubnets"
"ec2:DescribeSecurityGroups"
"ec2:CreateNetworkInterface"
```

Darüber hinaus muss Ihre IAM-Richtlinie auch die Erstellung der dienstbezogenen Rolle ermöglichen:

```
"iam:CreateServiceLinkedRole"
```

> **Note**
>
> Keine dieser Berechtigungen ist für Benutzer erforderlich, die in ihren Projekten keine VPC-Konfigurationen verwenden.

Service-verknüpfte Rollen

AWS Device Farm verwendet AWS Identity and Access Management (IAM) serviceverknüpfte Rollen. Eine dienstverknüpfte Rolle ist ein einzigartiger Typ von IAM-Rolle, die direkt mit Device Farm verknüpft ist. Dienstbezogene Rollen sind von Device Farm vordefiniert und enthalten alle Berechtigungen, die der Dienst benötigt, um andere AWS Dienste in Ihrem Namen aufzurufen.

Eine dienstverknüpfte Rolle erleichtert die Einrichtung von Device Farm, da Sie die erforderlichen Berechtigungen nicht manuell hinzufügen müssen. Device Farm definiert die Berechtigungen seiner

dienstbezogenen Rollen, und sofern nicht anders definiert, kann nur Device Farm seine Rollen übernehmen. Die definierten Berechtigungen umfassen die Vertrauens- und Berechtigungsrichtlinie. Diese Berechtigungsrichtlinie kann keinen anderen IAM-Entitäten zugewiesen werden.

Sie können eine serviceverknüpfte Rolle erst löschen, nachdem ihre verwandten Ressourcen gelöscht wurden. Dadurch werden Ihre Device Farm Farm-Ressourcen geschützt, da Sie die Zugriffsberechtigung für die Ressourcen nicht versehentlich entfernen können.

Informationen zu anderen Services, die serviceverknüpfte Rollen unterstützen, finden Sie unter AWS-Services, die mit IAM funktionieren. Suchen Sie nach den Services, für die Ja in der Spalte Serviceverknüpfte Rolle angegeben ist. Wählen Sie über einen Link Ja aus, um die Dokumentation zu einer serviceverknüpften Rolle für diesen Service anzuzeigen.

Dienstbezogene Rollenberechtigungen für Device Farm

Device Farm verwendet die serviceverknüpfte Rolle mit dem Namen AWSServiceRoleForDeviceFarm— Erlaubt Device Farm, in Ihrem Namen auf AWS-Ressourcen zuzugreifen.

Die AWSService RoleForDeviceFarm serviceverknüpfte Rolle vertraut darauf, dass die folgenden Dienste die Rolle übernehmen:

- `devicefarm.amazonaws.com`

Die Rollenberechtigungsrichtlinie ermöglicht es Device Farm, die folgenden Aktionen auszuführen:

- Für Ihr Konto
  - Netzwerkschnittstellen erstellen
  - Beschreiben von Netzwerkschnittstellen
  - Beschreiben VPCs
  - Beschreiben Sie Subnetze
  - Beschreiben von Sicherheitsgruppen
  - Schnittstellen löschen
  - Netzwerkschnittstellen ändern
- Für Netzwerkschnittstellen
  - Tags erstellen
- Für EC2-Netzwerkschnittstellen, die von Device Farm verwaltet werden

- Berechtigungen für Netzwerkschnittstellen erstellen

Die vollständige IAM-Richtlinie lautet:

JSON

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

Sie müssen Berechtigungen konfigurieren, damit eine juristische Stelle von IAM (z. B. Benutzer, Gruppe oder Rolle) eine serviceverknüpfte Rolle erstellen, bearbeiten oder löschen kann. Weitere Informationen finden Sie unter serviceverknüpfte Rollenberechtigungen im IAM-Benutzerhandbuch.

Eine dienstverknüpfte Rolle für Device Farm erstellen

Wenn Sie eine VPC-Konfiguration für ein mobiles Testprojekt bereitstellen, müssen Sie eine serviceverknüpfte Rolle nicht manuell erstellen. Wenn Sie Ihre erste Device Farm-Ressource in der AWS-Managementkonsole, der AWS CLI oder der AWS API erstellen, erstellt Device Farm die dienstverknüpfte Rolle für Sie.

Wenn Sie diese serviceverknüpfte Rolle löschen und sie dann erneut erstellen müssen, können Sie dasselbe Verfahren anwenden, um die Rolle in Ihrem Konto neu anzulegen. Wenn Sie Ihre erste Device Farm-Ressource erstellen, erstellt Device Farm die dienstverknüpfte Rolle erneut für Sie.

Sie können die IAM-Konsole auch verwenden, um eine dienstverknüpfte Rolle mit dem Anwendungsfall Device Farm zu erstellen. Erstellen Sie in der AWS CLI oder der AWS API eine dienstverknüpfte Rolle mit dem `devicefarm.amazonaws.com` Dienstnamen. Weitere Informationen finden Sie unter Erstellen einer serviceverknüpfte Rolle im IAM-Leitfaden. Wenn Sie diese serviceverknüpfte Rolle löschen, können Sie mit demselben Verfahren die Rolle erneut erstellen.

Bearbeiten einer dienstverknüpften Rolle für Device Farm

Device Farm erlaubt es Ihnen nicht, die AWSService RoleForDeviceFarm dienstverknüpfte Rolle zu bearbeiten. Da möglicherweise verschiedene Entitäten auf die Rolle verweisen, kann der Rollenname nach dem Erstellen einer serviceverknüpften Rolle nicht mehr geändert werden. Sie können jedoch die Beschreibung der Rolle mit IAM bearbeiten. Weitere Informationen finden Sie unter Bearbeiten einer serviceverknüpften Rolle im IAM-Benutzerhandbuch.

Löschen einer dienstverknüpften Rolle für Device Farm

Wenn Sie ein Feature oder einen Dienst, die bzw. der eine serviceverknüpften Rolle erfordert, nicht mehr benötigen, sollten Sie diese Rolle löschen. Auf diese Weise haben Sie keine ungenutzte juristische Stelle, die nicht aktiv überwacht oder verwaltet wird. Sie müssen jedoch die Ressourcen für Ihre serviceverknüpften Rolle zunächst bereinigen, bevor Sie sie manuell löschen können.

> **Note**
>
> Wenn der Device Farm Farm-Dienst die Rolle verwendet, wenn Sie versuchen, die Ressourcen zu löschen, schlägt das Löschen möglicherweise fehl. Wenn dies passiert, warten Sie einige Minuten und versuchen Sie es erneut.

So löschen Sie die serviceverknüpfte Rolle mit IAM

Verwenden Sie die IAM-Konsole, die oder die AWS API AWS CLI, um die AWSService RoleForDeviceFarm dienstverknüpfte Rolle zu löschen. Weitere Informationen finden Sie unter Löschen einer serviceverknüpften Rolle im IAM-Leitfaden.

Unterstützte Regionen für mit Device Farm Farm-Dienst verknüpfte Rollen

Device Farm unterstützt die Verwendung von dienstbezogenen Rollen in allen Regionen, in denen der Dienst verfügbar ist. Weitere Informationen finden Sie unter AWS -Regionen und Endpunkte.

Device Farm unterstützt nicht die Verwendung von dienstbezogenen Rollen in allen Regionen, in denen der Dienst verfügbar ist. Sie können die AWSService RoleForDeviceFarm Rolle in den folgenden Regionen verwenden.

Name der Region	Regions-ID	Support in Device Farm
USA Ost (Nord-Virginia)	us-east-1	Nein
USA Ost (Ohio)	us-east-2	Nein
USA West (Nordkalifornien)	us-west-1	Nein
USA West (Oregon)	us-west-2	Ja
Asien-Pazifik (Mumbai)	ap-south-1	Nein

Name der Region	Regions-ID	Support in Device Farm
Asia Pacific (Osaka)	ap-northeast-3	Nein
Asien-Pazifik (Seoul)	ap-northeast-2	Nein
Asien-Pazifik (Singapur)	ap-southeast-1	Nein
Asien-Pazifik (Sydney)	ap-southeast-2	Nein
Asien-Pazifik (Tokio)	ap-northeast-1	Nein
Kanada (Zentral)	ca-central-1	Nein
Europa (Frankfurt)	eu-central-1	Nein
Europa (Irland)	eu-west-1	Nein
Europa (London)	eu-west-2	Nein
Europa (Paris)	eu-west-3	Nein
Südamerika (São Paulo)	sa-east-1	Nein
AWS GovCloud (US)	us-gov-west-1	Nein

Voraussetzungen

In der folgenden Liste werden einige Anforderungen und Vorschläge beschrieben, die Sie bei der Erstellung von VPC-ENI-Konfigurationen beachten sollten:

- Private Geräte müssen Ihrem AWS Konto zugewiesen werden.
- Sie müssen über einen AWS Kontobenutzer oder eine Rolle mit den erforderlichen Berechtigungen verfügen, um eine mit dem Dienst verknüpfte Rolle zu erstellen. Bei der Verwendung von Amazon VPC-Endpunkten mit den mobilen Testfunktionen von Device Farm erstellt Device Farm eine AWS Identity and Access Management (IAM) -Serviceverknüpfte Rolle.
- Device Farm kann VPCs nur in der `us-west-2` Region eine Verbindung herstellen. Wenn Sie in der `us-west-2` Region keine VPC haben, müssen Sie eine erstellen. Um dann auf Ressourcen in einer VPC in einer anderen Region zuzugreifen, müssen Sie eine Peering-Verbindung zwischen

der VPC in der `us-west-2` Region und der VPC in der anderen Region herstellen. Informationen zum Peering VPCs finden Sie im Amazon VPC Peering Guide.

Sie sollten bei der Konfiguration der Verbindung sicherstellen, dass Sie Zugriff auf Ihre angegebene VPC haben. Sie müssen bestimmte Amazon Elastic Compute Cloud (Amazon EC2) - Berechtigungen für Device Farm konfigurieren.

- In der VPC, die Sie verwenden, ist eine DNS-Auflösung erforderlich.
- Sobald Ihre VPC erstellt wurde, benötigen Sie die folgenden Informationen über die VPC in der `us-west-2` Region:
  - VPC-ID
  - Subnetz IDs (nur private Subnetze)
  - Sicherheitsgruppe IDs
- Sie müssen Amazon VPC-Verbindungen pro Projekt konfigurieren. Derzeit können Sie nur eine VPC-Konfiguration pro Projekt konfigurieren. Wenn Sie eine VPC konfigurieren, erstellt Amazon VPC eine Schnittstelle innerhalb Ihrer VPC und weist sie den angegebenen Subnetzen und Sicherheitsgruppen zu. Alle future Sitzungen, die mit dem Projekt verknüpft sind, werden die konfigurierte VPC-Verbindung verwenden.
- Sie können VPC-ENI-Konfigurationen nicht zusammen mit der älteren VPCE-Funktion verwenden.
- Wir empfehlen dringend, ein vorhandenes Projekt nicht mit einer VPC-ENI-Konfiguration zu aktualisieren, da bestehende Projekte möglicherweise VPCE-Einstellungen haben, die auf der Run-Ebene bestehen bleiben. Wenn Sie die vorhandenen VPCE-Funktionen bereits verwenden, verwenden Sie stattdessen VPC-ENI für alle neuen Projekte.

Verbindung zu Amazon VPC herstellen

Sie können Ihr Projekt für die Verwendung von Amazon VPC-Endpunkten konfigurieren und aktualisieren. Die VPC-ENI-Konfiguration wird pro Projekt konfiguriert. Ein Projekt kann zu einem bestimmten Zeitpunkt nur einen VPC-ENI-Endpunkt haben. Um den VPC-Zugriff für ein Projekt zu konfigurieren, müssen Sie die folgenden Details kennen:

- Die VPC-ID in, `us-west-2` wenn Ihre App dort gehostet wird, oder die `us-west-2` VPC-ID, die eine Verbindung zu einer anderen VPC in einer anderen Region herstellt.
- Die entsprechenden Sicherheitsgruppen, die auf die Verbindung angewendet werden sollen.

- Die Subnetze, die der Verbindung zugeordnet werden. Wenn eine Sitzung gestartet wird, wird das größte verfügbare Subnetz verwendet. Wir empfehlen, mehrere Subnetze verschiedenen Availability Zones zuzuordnen, um die Verfügbarkeit Ihrer VPC-Konnektivität zu verbessern.
- Bei Verwendung von VPC-ENI ist der von den Device Farm Farm-Testhosts und -Geräten verwendete DNS-Resolver der Server, der von den DHCP-Diensten im Kundensubnetz bereitgestellt wird. In einer Standardkonfiguration ist dies der Standard-Resolver der VPC. Kunden, die benutzerdefinierte DNS-Resolver angeben möchten, können in ihrer VPC einen DHCP-Optionssatz konfigurieren.

Nachdem Sie Ihre VPC-ENI-Konfiguration erstellt haben, können Sie ihre Details mithilfe der Konsole oder CLI wie folgt aktualisieren.

Console

1. Melden Sie sich bei der Device Farm Farm-Konsole unter https://console.aws.amazon.com/devicefarm an.
2. Wählen Sie im Navigationsbereich Device Farm die Option Mobile Device Testing und dann Projects aus.
3. Wählen Sie unter Mobile Testing-Projekte den Namen Ihres Projekts aus der Liste aus.
4. Wählen Sie Project settings (Projekteinstellungen) aus.
5. Im Abschnitt Virtual Private Cloud (VPC) -Einstellungen können SieVPC, `Subnets` (nur private Subnetze) und ändern. `Security Groups`
6. Wählen Sie Speichern.

CLI

Verwenden Sie den folgenden AWS-CLI-Befehl, um die Amazon VPC zu aktualisieren:

```
$  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
```

Sie können bei der Erstellung Ihres Projekts auch eine Amazon VPC konfigurieren:

```
$  aws devicefarm create-project \
--name VPCDemo \
--vpc-config \
securityGroupIds=sg-02c1537701a7e3763,sg-005dadf9311efda25,\
subnetIds=subnet-09b1a45f9cac53717,subnet-09b1a45f9cac12345,\
vpcId=vpc-0238fb322af81a368
```

Einschränkungen

Die folgenden Einschränkungen gelten für die VPC-ENI-Funktion:

- Sie können bis zu fünf Sicherheitsgruppen in der VPC-Konfiguration eines Device Farm Farm-Projekts angeben.
- Sie können bis zu acht Subnetze in der VPC-Konfiguration eines Device Farm Farm-Projekts bereitstellen.
- Wenn Sie ein Device Farm Farm-Projekt für die Verwendung mit Ihrer VPC konfigurieren, muss das kleinste Subnetz, das Sie bereitstellen können, mindestens fünf verfügbare IPv4 Adressen haben.
- Öffentliche IP-Adressen werden derzeit nicht unterstützt. Stattdessen empfehlen wir, private Subnetze in Ihren Device Farm Farm-Projekten zu verwenden. Wenn Sie während Ihrer Tests einen öffentlichen Internetzugang benötigen, verwenden Sie ein NAT-Gateway (Network Address Translation). Durch die Konfiguration eines Device Farm Farm-Projekts mit einem öffentlichen Subnetz erhalten Ihre Tests weder Internetzugang noch eine öffentliche IP-Adresse.
- Die VPC-ENI-Integration unterstützt nur private Subnetze in Ihrer VPC.
- Nur ausgehender Datenverkehr von der dienstverwalteten ENI wird unterstützt. Dies bedeutet, dass die ENI keine unaufgeforderten eingehenden Anfragen von der VPC empfangen kann.

Verwendung von Amazon VPC-Endpunktservices mit Device Farm — Legacy (nicht empfohlen)

> **Warning**
>
> Wir empfehlen dringend, die auf dieser Seite beschriebene VPC-ENI-Konnektivität für private Endpunktkonnektivität zu verwenden, da VPCE jetzt als Legacy-Funktion gilt. VPC-ENI bietet

mehr Flexibilität, einfachere Konfigurationen, ist kostengünstiger und erfordert im Vergleich zur VPCE-Konnektivitätsmethode deutlich weniger Wartungsaufwand.

> **Note**
>
> Die Verwendung von Amazon VPC Endpoint Services mit Device Farm wird nur für Kunden mit konfigurierten privaten Geräten unterstützt. Um Ihrem AWS-Konto die Nutzung dieser Funktion mit privaten Geräten zu ermöglichen, kontaktieren Sie uns bitte.

Amazon Virtual Private Cloud (Amazon VPC) ist ein AWS Service, mit dem Sie AWS Ressourcen in einem von Ihnen definierten virtuellen Netzwerk starten können. Mit einer VPC haben Sie die Kontrolle über Ihre Netzwerkeinstellungen, z. B. den IP-Adressbereich, Subnetze, Routingtabellen und Netzwerk-Gateways.

Wenn Sie Amazon VPC verwenden, um private Anwendungen in der AWS Region USA West (Oregon) (`us-west-2`) zu hosten, können Sie eine private Verbindung zwischen Ihrer VPC und Device Farm herstellen. Mit dieser Verbindung können Sie Device Farm verwenden, um private Anwendungen zu testen, ohne sie über das öffentliche Internet verfügbar zu machen. Wenden Sie sich an uns, damit Ihr AWS Konto diese Funktion auf privaten Geräten nutzen kann.

Um eine Ressource in Ihrer VPC mit Device Farm zu verbinden, können Sie die Amazon VPC-Konsole verwenden, um einen VPC-Endpunktservice zu erstellen. Mit diesem Endpunktdienst können Sie die Ressource in Ihrer VPC über einen Device Farm-VPC-Endpunkt für die Device Farm bereitstellen. Der Endpunktdienst bietet zuverlässige, skalierbare Konnektivität zu Device Farm, ohne dass ein Internet-Gateway, eine NAT-Instanz (Network Address Translation) oder eine VPN-Verbindung erforderlich ist. Weitere Informationen finden Sie unter VPC Endpoint Services (AWS PrivateLink) im AWS PrivateLink Handbuch.

> **Important**
>
> Die VPC-Endpunktfunktion von Device Farm hilft Ihnen dabei, private interne Dienste in Ihrer VPC mithilfe von Verbindungen sicher mit der öffentlichen Device Farm Farm-VPC zu verbinden. AWS PrivateLink Obwohl die Verbindung sicher und privat ist, hängt die Sicherheit vom Schutz Ihrer AWS -Anmeldeinformationen ab. Wenn Ihre AWS Anmeldeinformationen

kompromittiert werden, kann ein Angreifer auf Ihre Servicedaten zugreifen oder diese der Außenwelt zugänglich machen.

Nachdem Sie einen VPC-Endpunkt-Service in Amazon VPC erstellt haben, können Sie die Device Farm-Konsole verwenden, um eine VPC-Endpunktkonfiguration in Device Farm zu erstellen. In diesem Thema erfahren Sie, wie Sie die Amazon VPC-Verbindung und die VPC-Endpunktkonfiguration in Device Farm erstellen.

Bevor Sie beginnen

Die folgenden Informationen gelten für Amazon VPC-Benutzer in der Region USA West (Oregon) (`us-west-2`) mit einem Subnetz in jeder der folgenden Availability Zones: us-west-2a, us-west-2b und us-west-2c.

Device Farm stellt zusätzliche Anforderungen an die VPC-Endpunktdienste, mit denen Sie es verwenden können. Wenn Sie einen VPC-Endpunktdienst für die Verwendung mit Device Farm erstellen und konfigurieren, stellen Sie sicher, dass Sie Optionen auswählen, die die folgenden Anforderungen erfüllen:

- Die Availability Zones für den Service müssen us-west-2a, us-west-2b und us-west-2c umfassen. Der Network Load Balancer, der einem VPC-Endpunktdienst zugeordnet ist, bestimmt die Availability Zones für diesen VPC-Endpunktdienst. Wenn Ihr VPC-Endpunktdienst nicht alle drei Availability Zones anzeigt, müssen Sie Ihren Network Load Balancer neu erstellen, um diese drei Zonen zu aktivieren, und dann den Network Load Balancer wieder Ihrem Endpunktdienst zuordnen.
- Die zulässigen Prinzipale für den Endpunkt-Service müssen den Amazon-Ressourcennamen (ARN) des Device Farm Farm-VPC-Endpunkts (Service-ARN) enthalten. Nachdem Sie Ihren Endpunktdienst erstellt haben, fügen Sie den Device Farm VPC-Endpunktdienst-ARN zu Ihrer Zulassungsliste hinzu, um Device Farm die Erlaubnis zu erteilen, auf Ihren VPC-Endpunktdienst zuzugreifen. Kontaktieren Sie uns, um den Device Farm VPC-Endpunktdienst ARN zu erhalten.

Wenn Sie bei der Erstellung Ihres VPC-Endpunktdienstes die Einstellung Akzeptanz erforderlich aktiviert lassen, müssen Sie außerdem jede Verbindungsanforderung, die Device Farm an den Endpunktdienst sendet, manuell akzeptieren. Um diese Einstellung für einen vorhandenen Endpunkt-Service zu ändern, wählen Sie den Endpunkt-Service auf der Amazon VPC-Konsole aus, wählen Sie Aktionen und dann Endpunkt-Akzeptanzeinstellung ändern. Weitere Informationen finden Sie im Leitfaden unter Ändern der Load Balancer- und Akzeptanzeinstellungen.AWS PrivateLink

Im nächsten Abschnitt wird erklärt, wie Sie einen Amazon VPC-Endpunktservice erstellen, der diese Anforderungen erfüllt.

Schritt 1: Einen Network Load Balancer erstellen

Der erste Schritt beim Aufbau einer privaten Verbindung zwischen Ihrer VPC und Device Farm besteht darin, einen Network Load Balancer einzurichten, der Anfragen an eine Zielgruppe weiterleitet.

New console

So erstellen Sie einen Network Load Balancer mit der neuen Konsole

1. Öffnen Sie die Amazon Elastic Compute Cloud (Amazon EC2) -Konsole unter https://console.aws.amazon.com/ec2/.
2. Wählen Sie im Navigationsbereich unter Load Balancing die Option Load Balancers aus.
3. Wählen Sie Load Balancer erstellen aus.
4. Wählen Sie unter Network Load Balancer die Option Create aus.
5. Gehen Sie auf der Seite Network Load Balancer erstellen unter Grundkonfiguration wie folgt vor:
   a. Geben Sie einen Namen für den Load Balancer ein.
   b. Wählen Sie für Schema die Option Intern aus.
6. Führen Sie unter Network mapping (Netzwerk-Mapping) einen der folgenden Schritte aus:
   a. Wählen Sie die VPC für Ihre Zielgruppe.
   b. Wählen Sie die folgenden Mappings aus:
      - `us-west-2a`
      - `us-west-2b`
      - `us-west-2c`
7. Verwenden Sie unter Listener und Routing die Optionen Protokoll und Port, um Ihre Zielgruppe auszuwählen.

> **Note**
>
> Standardmäßig ist der zonenübergreifende Lastenausgleich deaktiviert.

Da der Load Balancer die Availability Zones verwendet `us-west-2aus-west-2b`, erfordert er entweder`us-west-2c`, dass Ziele in jeder dieser Availability Zones registriert sind, oder, wenn Sie Ziele in weniger als allen drei Zonen registrieren, müssen Sie zonenübergreifendes Load Balancing aktivieren. Andernfalls funktioniert der Load Balancer möglicherweise nicht wie erwartet.

8. Wählen Sie Load Balancer erstellen aus.

Old console

So erstellen Sie einen Network Load Balancer mit der alten Konsole

1. Öffnen Sie die Amazon Elastic Compute Cloud (Amazon EC2) -Konsole unter https://console.aws.amazon.com/ec2/.
2. Wählen Sie im Navigationsbereich unter Load Balancing die Option Load Balancers aus.
3. Wählen Sie Load Balancer erstellen aus.
4. Wählen Sie unter Network Load Balancer die Option Create aus.
5. Gehen Sie auf der Seite Load Balancer konfigurieren unter Grundkonfiguration wie folgt vor:
   a. Geben Sie einen Namen für den Load Balancer ein.
   b. Wählen Sie für Schema die Option Intern aus.
6. Wählen Sie unter Listeners das Protokoll und den Port aus, die Ihre Zielgruppe verwendet.
7. Gehen Sie unter Availability Zones wie folgt vor:
   a. Wählen Sie die VPC für Ihre Zielgruppe.
   b. Wählen Sie die folgenden Availability Zones aus:
      - `us-west-2a`
      - `us-west-2b`
      - `us-west-2c`
   c. Wählen Sie Weiter: Sicherheitseinstellungen konfigurieren.
8. (Optional) Konfigurieren Sie Ihre Sicherheitseinstellungen und wählen Sie dann Weiter: Routing konfigurieren.
9. Führen Sie auf der Seite Configure Routing (Weiterleitung konfigurieren) die folgenden Schritte aus:

a. Für Zielgruppe, wählen Sie Vorhandene Zielgruppe.

b. Wählen Sie unter Name Ihre Zielgruppe aus.

c. Wählen Sie Weiter: Ziele registrieren.

10. Überprüfen Sie auf der Seite Ziele registrieren Ihre Ziele und wählen Sie dann Weiter: überprüfen aus.

> **Note**
>
> Standardmäßig ist der zonenübergreifende Load Balancing deaktiviert.
> Da der Load Balancer die Availability Zones verwendet `us-west-2aus-west-2b`, erfordert er entweder`us-west-2c`, dass Ziele in jeder dieser Availability Zones registriert sind, oder, wenn Sie Ziele in weniger als allen drei Zonen registrieren, müssen Sie zonenübergreifendes Load Balancing aktivieren. Andernfalls funktioniert der Load Balancer möglicherweise nicht wie erwartet.

11. Überprüfen Sie Ihre Load Balancer-Konfiguration und wählen Sie dann Create aus.

Schritt 2: Einen Amazon VPC-Endpunktservice erstellen

Nachdem Sie den Network Load Balancer erstellt haben, verwenden Sie die Amazon VPC-Konsole, um einen Endpunkt-Service in Ihrer VPC zu erstellen.

1. Öffnen Sie die Amazon-VPC-Konsole unter https://console.aws.amazon.com/vpc/.
2. Wählen Sie unter Ressourcen nach Region die Option Endpoint Services aus.
3. Wählen Sie Create Endpoint Service (Endpunkt-Service erstellen) aus.
4. Führen Sie eine der folgenden Aktionen aus:

   - Wenn Sie bereits über einen Network Load Balancer verfügen, den der Endpoint Service verwenden soll, wählen Sie ihn unter Verfügbare Load Balancer aus, und fahren Sie dann mit Schritt 5 fort.
   - Wenn Sie noch keinen Network Load Balancer erstellt haben, wählen Sie Create new Load Balancer aus. Die Amazon EC2 EC2-Konsole wird geöffnet. Folgen Sie ab Schritt 3 den Schritten unter Network Load Balancer erstellen und fahren Sie dann mit diesen Schritten in der Amazon VPC-Konsole fort.

5. Vergewissern Sie sich, dass bei Inbegriffene Availability Zones `us-west-2aus-west-2b`, dass, und in der Liste `us-west-2c` erscheinen.
6. Wenn Sie nicht jede Verbindungsanfrage, die an den Endpoint Service gesendet wird, manuell annehmen oder ablehnen möchten, deaktivieren Sie unter Zusätzliche Einstellungen die Option Annahme erforderlich. Wenn Sie diese Option deaktivieren, akzeptiert der Endpunkt-Service automatisch alle Verbindungsanforderungen, die er empfängt.
7. Wählen Sie Erstellen aus.
8. Wählen Sie im neuen Endpunktdienst die Option Prinzipale zulassen aus.
9. Kontaktieren Sie uns, um den ARN des Device Farm Farm-VPC-Endpunkts (Dienst-ARN) zu erhalten, der der Zulassungsliste für den Endpunktdienst hinzugefügt werden soll, und fügen Sie diesen Dienst-ARN dann der Zulassungsliste für den Dienst hinzu.
10. Notieren Sie sich auf der Registerkarte Details des Endpunkt-Services, den Namen des Services (Service-Name). Sie benötigen diesen Namen, wenn Sie die VPC-Endpunkt-Konfiguration im nächsten Schritt erstellen.

Ihr VPC-Endpunktdienst kann jetzt mit Device Farm verwendet werden.

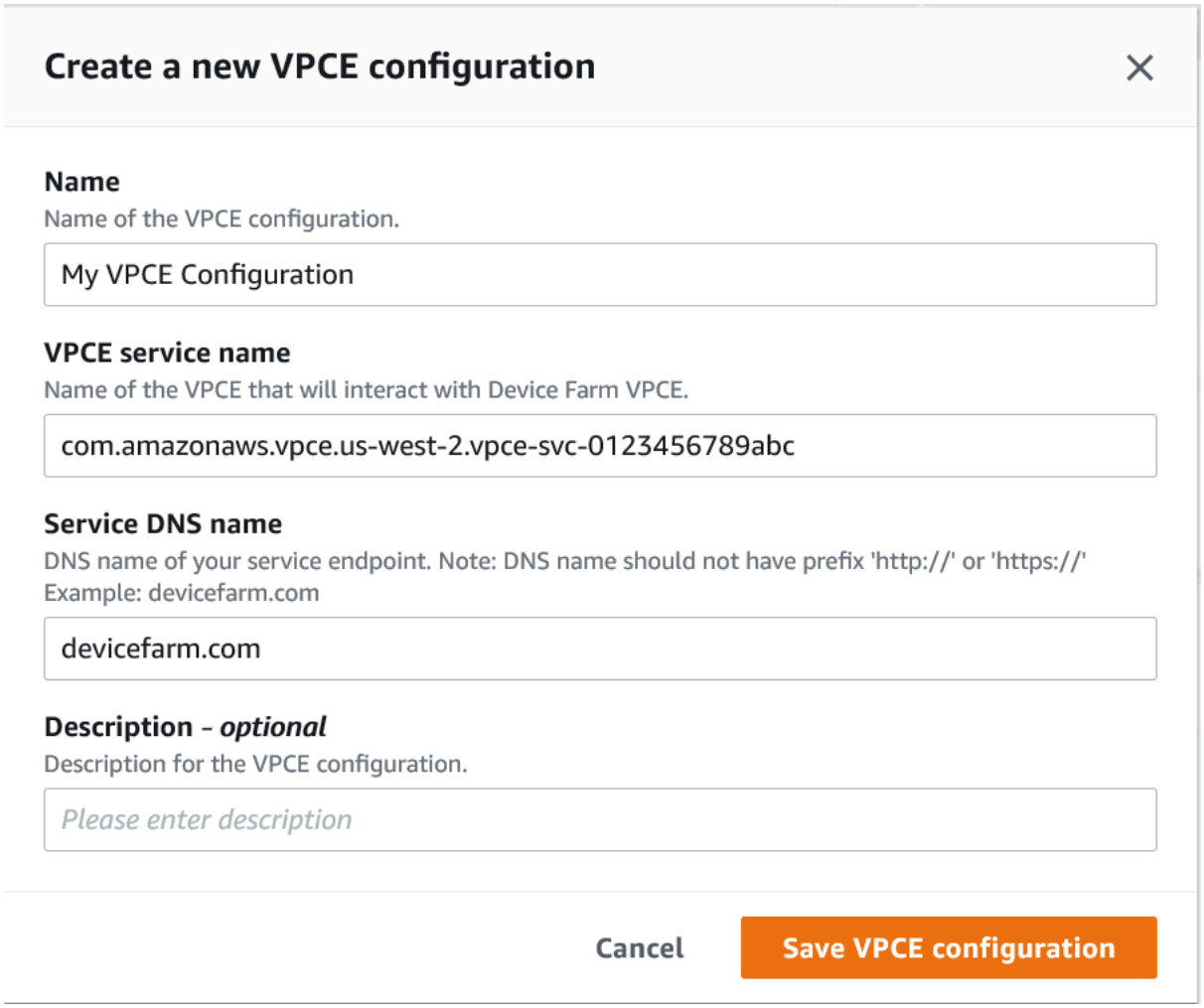
Schritt 3: Erstellen einer VPC-Endpunktkonfiguration in Device Farm

Nachdem Sie einen Endpunkt-Service in Amazon VPC erstellt haben, können Sie in Device Farm eine Amazon VPC-Endpunktkonfiguration erstellen.

1. Melden Sie sich bei der Device Farm Farm-Konsole unter https://console.aws.amazon.com/devicefarm an.
2. Wählen Sie im Navigationsbereich die Option Testen von Mobilgeräten und dann Private Geräte aus.
3. Wählen Sie VPCE-Konfigurationen aus.
4. Wählen Sie VPCE-Konfiguration erstellen.
5. Geben Sie unter Neue VPCE-Konfiguration erstellen einen Namen für die VPC-Endpunktkonfiguration ein.
6. Geben Sie als VPCE-Servicename den Namen des Amazon VPC-Endpunktdienstes (Servicename) ein, den Sie in der Amazon VPC-Konsole notiert haben. Das Name sieht wie folgt aus: `com.amazonaws.vpce.us-west-2.vpce-svc-id`.

7. Geben Sie unter Service-DNS-Name den Service-DNS-Namen für die App ein, die Sie testen möchten (z. B.). `devicefarm.com` Geben Sie vor dem Service-DNS-Namen weder `http` noch `https` an.

   Der Domänenname ist nicht über das öffentliche Internet verfügbar. Darüber hinaus wird dieser neue Domainname, der Ihrem VPC-Endpunktservice zugeordnet ist, von Amazon Route 53 generiert und steht Ihnen in Ihrer Device Farm Farm-Sitzung exklusiv zur Verfügung.
8. Wählen Sie Speichern.



Schritt 4: Einen Testlauf erstellen

Nachdem Sie die VPC-Endpunktkonfiguration gespeichert haben, können Sie die Konfiguration verwenden, um Testläufe oder Fernzugriffssitzungen zu erstellen. Für weitere Informationen siehe Einen Testlauf in Device Farm erstellen oder Erstellen einer Sitzung.

Protokollieren von API-Aufrufen von AWS Device Farm mit AWS CloudTrail

AWS Device Farm ist in einen Service integriert AWS CloudTrail, der eine Aufzeichnung der Aktionen bereitstellt, die von einem Benutzer, einer Rolle oder einem AWS Service in AWS Device Farm ausgeführt wurden. CloudTrail erfasst alle API-Aufrufe für AWS Device Farm als Ereignisse. Zu den erfassten Aufrufen gehören Aufrufe von der AWS Device Farm-Konsole und Codeaufrufen für die API-Operationen von AWS Device Farm. Wenn Sie einen Trail erstellen, können Sie die kontinuierliche Bereitstellung von CloudTrail Ereignissen an einen Amazon S3 S3-Bucket aktivieren, einschließlich Ereignissen für AWS Device Farm. Wenn Sie keinen Trail konfigurieren, können Sie die neuesten Ereignisse trotzdem in der CloudTrail Konsole im Ereignisverlauf anzeigen. Anhand der von gesammelten Informationen können Sie die Anfrage CloudTrail, die an AWS Device Farm gestellt wurde, die IP-Adresse, von der aus die Anfrage gestellt wurde, wer die Anfrage gestellt hat, wann sie gestellt wurde, und weitere Details ermitteln.

Weitere Informationen CloudTrail dazu finden Sie im AWS CloudTrail Benutzerhandbuch.

Informationen zur AWS-Gerätefarm in CloudTrail

CloudTrail ist für Ihr AWS Konto aktiviert, wenn Sie das Konto erstellen. Wenn eine Aktivität in AWS Device Farm auftritt, wird diese Aktivität zusammen mit anderen AWS Serviceereignissen im CloudTrail Ereignisverlauf in einem Ereignis aufgezeichnet. Sie können aktuelle Ereignisse in Ihrem AWS Konto anzeigen, suchen und herunterladen. Weitere Informationen finden Sie unter Ereignisse mit CloudTrail Ereignisverlauf anzeigen.

Für eine fortlaufende Aufzeichnung von Ereignissen in Ihrem AWS Konto, einschließlich Ereignissen für AWS Device Farm, erstellen Sie einen Trail. Ein Trail ermöglicht CloudTrail die Übermittlung von Protokolldateien an einen Amazon S3 S3-Bucket. Wenn Sie einen Pfad in der Konsole anlegen, gilt dieser für alle AWS-Regionen. Der Trail protokolliert Ereignisse aus allen Regionen der AWS Partition und übermittelt die Protokolldateien an den von Ihnen angegebenen Amazon S3 S3-Bucket. Darüber hinaus können Sie andere AWS Dienste konfigurieren, um die in den CloudTrail Protokollen gesammelten Ereignisdaten weiter zu analysieren und darauf zu reagieren. Weitere Informationen finden Sie hier:

- Übersicht zum Erstellen eines Trails
- CloudTrail Unterstützte Dienste und Integrationen

- Konfiguration von Amazon SNS SNS-Benachrichtigungen für CloudTrail
- Empfangen von CloudTrail Protokolldateien aus mehreren Regionen und Empfangen von CloudTrail Protokolldateien von mehreren Konten

Wenn die CloudTrail Protokollierung in Ihrem AWS Konto aktiviert ist, werden API-Aufrufe von Device Farm Farm-Aktionen in Protokolldateien aufgezeichnet. Device Farm Farm-Datensätze werden zusammen mit anderen AWS Dienstaufzeichnungen in eine Protokolldatei geschrieben. CloudTrail bestimmt anhand eines Zeitraums und der Dateigröße, wann eine neue Datei erstellt und in diese geschrieben werden soll.

Alle Device Farm Farm-Aktionen werden in der und der protokolliert AWS CLI Referenz und dokumentiertDevice Farm automatisieren. Aufrufe zum Erstellen eines neuen Projekts oder zur Ausführung in Device Farm generieren beispielsweise Einträge in CloudTrail Protokolldateien.

Jeder Ereignis- oder Protokolleintrag enthält Informationen zu dem Benutzer, der die Anforderung generiert hat. Die Identitätsinformationen unterstützen Sie bei der Ermittlung der folgenden Punkte:

- Ob die Anfrage mit Root- oder AWS Identity and Access Management (IAM-) Benutzeranmeldedaten gestellt wurde.
- Gibt an, ob die Anforderung mit temporären Sicherheitsanmeldeinformationen für eine Rolle oder einen Verbundbenutzer gesendet wurde.
- Ob die Anfrage von einem anderen AWS Dienst gestellt wurde.

Weitere Informationen finden Sie unter CloudTrail userIdentity-Element.

Grundlegendes zu Protokolldateieinträgen von AWS Device Farm

Ein Trail ist eine Konfiguration, die die Übertragung von Ereignissen als Protokolldateien an einen von Ihnen angegebenen Amazon S3 S3-Bucket ermöglicht. CloudTrail Protokolldateien enthalten einen oder mehrere Protokolleinträge. Ein Ereignis stellt eine einzelne Anforderung aus einer beliebigen Quelle dar und enthält Informationen über die angeforderte Aktion, Datum und Uhrzeit der Aktion, Anforderungsparameter usw. CloudTrail Protokolldateien sind kein geordneter Stack-Trace der öffentlichen API-Aufrufe, sodass sie nicht in einer bestimmten Reihenfolge angezeigt werden.

Das folgende Beispiel zeigt einen CloudTrail Protokolleintrag, der die `ListRuns` Aktion Device Farm demonstriert:

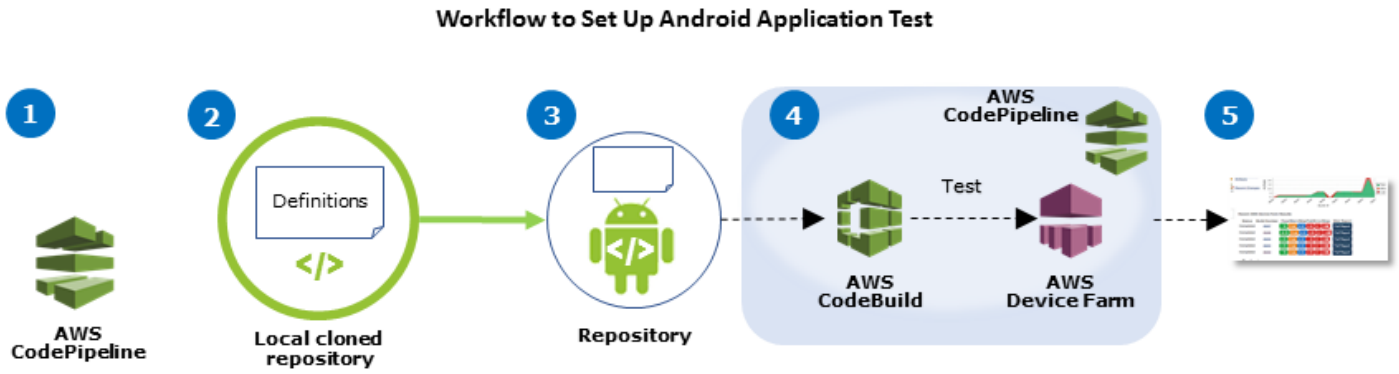
```
{
  "Records": [
    {
      "eventVersion": "1.03",
      "userIdentity": {
        "type": "Root",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:root",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
          "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 ...
          ]
        }
      }
    }
  ]
}
```

Integration von AWS Device Farm in einer CodePipeline Testphase

Sie können AWS CodePipelinees verwenden, um in Device Farm konfigurierte Tests für mobile Apps in eine AWS-verwaltete automatisierte Release-Pipeline zu integrieren. Sie können Ihre Pipeline so konfigurieren, dass Tests auf Aufforderung durchgeführt werden, nach einem festen Zeitplan oder als Teil eines stetigen Integrationsverlaufs.

Das folgende Diagramm zeigt den stetigen Integrationsverlauf, bei dem jedes Mal eine Android-App erstellt und getestet wird, wenn ein Push für ihr Repository bestätigt wird. Informationen zum Erstellen dieser Pipeline-Konfiguration finden Sie im Tutorial: Erstellen und Testen einer Android-App bei Push-Zugriff. GitHub



1. Konfiguration	2. Definitionen hinzufügen	3. Push	4. Bauen und testen	5. Bericht
Pipeline-Ressourcen konfigurieren	Ihrem Paket Build- und Testdefinitionen hinzufügen	Ein Paket in Ihr Repository übertragen	Automatisch ausgelöstes Erstellen und Testen von Apps aus Build-Ausgabeartefakten	Testergebnisse anzeigen

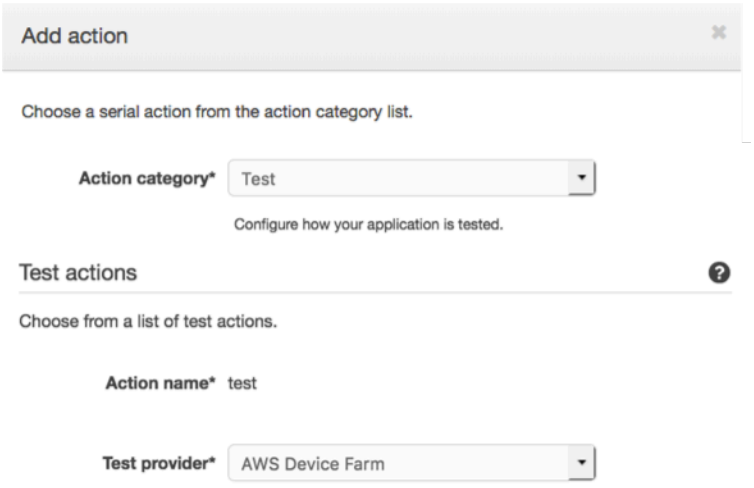
Informationen zum Konfigurieren einer Pipeline, die ständig Tests einer kompilierten App (beispielsweise eine iOS-`.ipa`- oder Android-`.apk`-Datei) als deren Quelle durchführt, finden Sie unter Tutorial: Testen einer iOS-App bei jedem Upload einer .ipa-Datei in einen Amazon S3-Bucket.

Für CodePipeline die Verwendung Ihrer Device Farm Farm-Tests konfigurieren

In diesen Schritten gehen wir davon aus, dass Sie ein Device Farm Farm-Projekt konfiguriert und eine Pipeline erstellt haben. Die Pipeline sollte mit einer Testphase konfiguriert sein, die ein Eingangsartefakt erhält, das Ihre Testdefinition und kompilierte App-Paketdateien beinhaltet. Das Eingangsartefakt der Testphase kann das Ausgangsartefakt einer in Ihrer Pipeline konfigurierten Quell- oder Build-Phase sein.

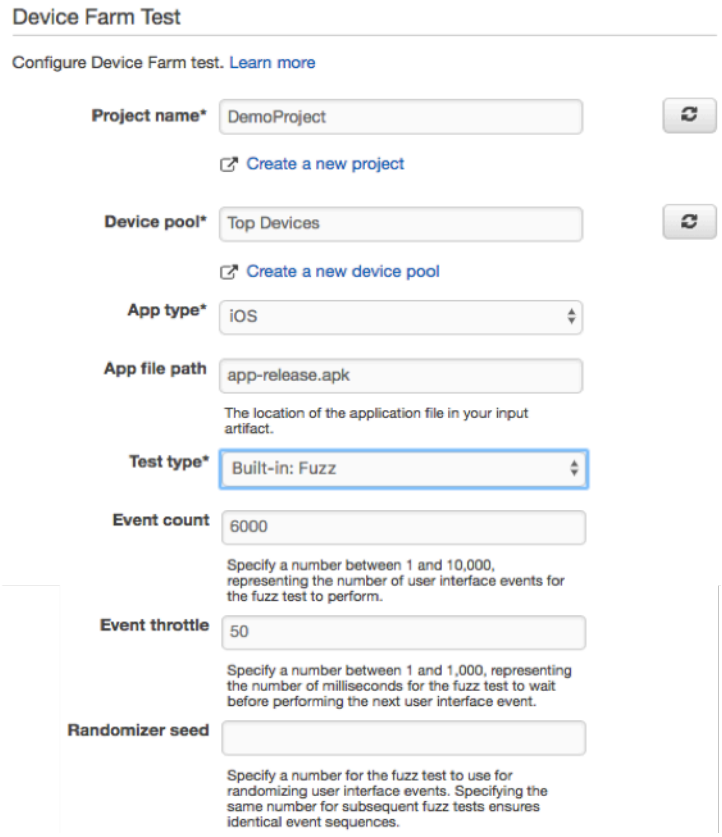
So konfigurieren Sie einen Device Farm Farm-Testlauf als CodePipeline Testaktion

1. Melden Sie sich bei der an AWS-Managementkonsole und öffnen Sie die CodePipeline Konsole unter https://console.aws.amazon.com/codepipeline/.
2. Wählen Sie die Pipeline für Ihre App-Version aus.
3. Klicken Sie im Feld für die Testphase auf das Bleistiftsymbol und wählen Sie dann Action (Aktion) aus.
4. Klicken Sie im Bereich Add action (Aktion hinzufügen) für Action category (Aktionskategorie) auf Test (Testen).
5. Geben Sie unter Action name (Aktionsname) einen Namen ein.
6. Wählen Sie unter Test provider (Testanbieter) die Option AWS Device Farm aus.



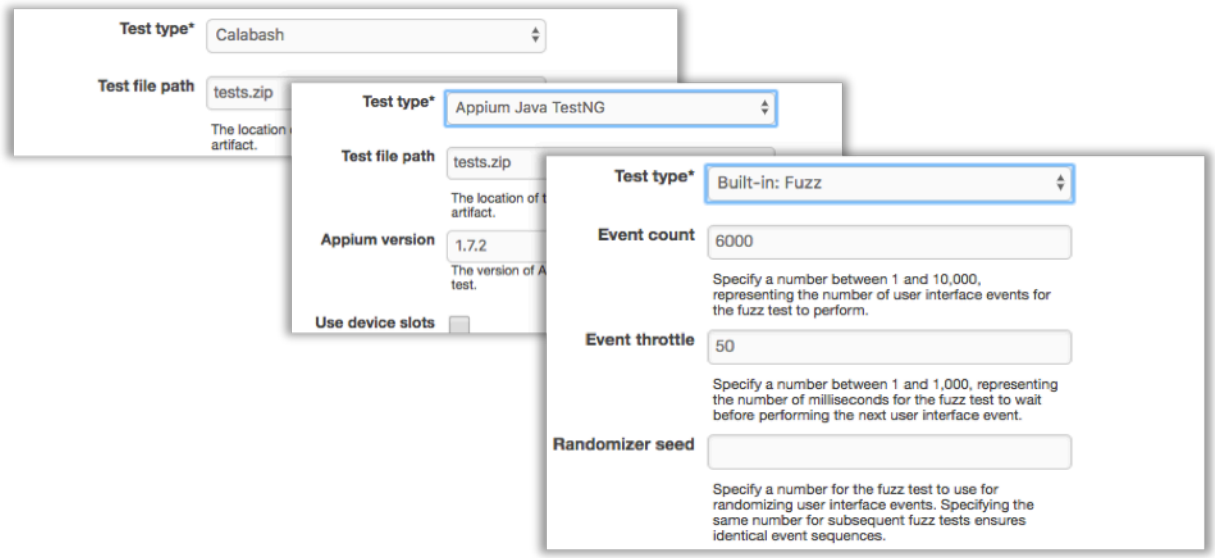
7. Wählen Sie unter Projektname Ihr vorhandenes Device Farm Farm-Projekt aus oder wählen Sie Neues Projekt erstellen aus.

8. Wählen Sie unter Device pool (Gerätepool) Ihren vorhandenen Gerätepool oder wählen Sie Create a new device pool (Einen neuen Gerätepool erstellen) aus. Wenn Sie einen Gerätepool erstellen, müssen Sie einen Satz Testgeräte auswählen.
9. Wählen Sie unter App type (App-Typ) die Plattform für Ihre App aus.

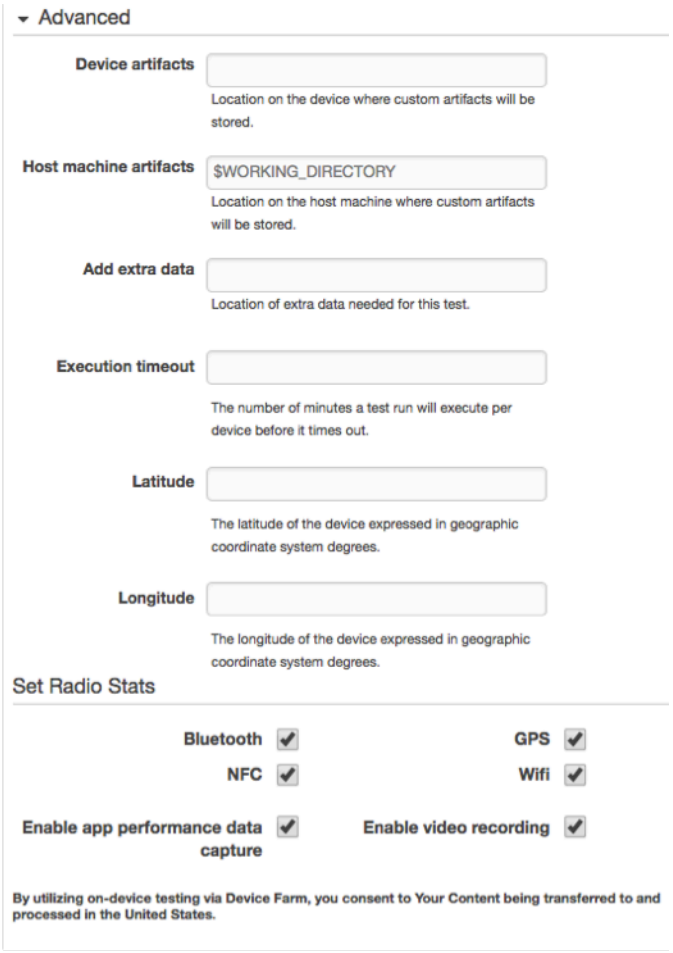


10. Geben Sie unter App file path (App-Dateipfad) den Pfad des kompilierten App-Pakets ein. Der Pfad ist relativ zum Stamm des Eingabeartefakts für Ihren Test.
11. Führen Sie unter Test type (Testtyp) einen der folgenden Schritte aus:

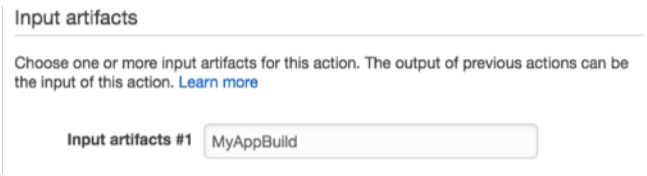
    - Wenn Sie einen der integrierten Device Farm Farm-Tests verwenden, wählen Sie den in Ihrem Device Farm Farm-Projekt konfigurierten Testtyp aus.
    - Wenn Sie keinen der integrierten Tests von Device Farm verwenden, geben Sie im Testdateipfad den Pfad der Testspezifikationsdatei ein. Der Pfad ist relativ zum Stamm des Eingabeartefakts für Ihren Test.



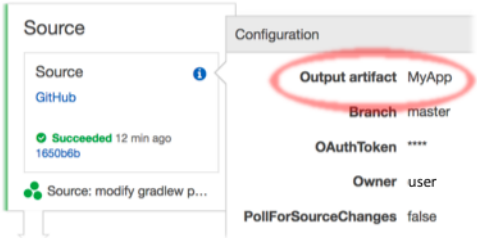
12. Geben Sie in den übrigen Feldern die Konfiguration ein, die für Ihren Test- und Anwendungstyp geeignet ist.
13. (Optional) Geben Sie unter Advanced (Erweitert) eine detaillierte Konfiguration für Ihren Testlauf ein.



14. Wählen Sie in Input artifacts (Eingangsartefakte) das Eingangsartefakt aus, das mit dem Ausgangsartefakt der Phase übereinstimmt, die sich vor der Testphase in der Pipeline befindet.



In der CodePipeline Konsole finden Sie den Namen des Ausgabeartefakts für jede Phase, indem Sie den Mauszeiger über das Informationssymbol im Pipeline-Diagramm bewegen. Wenn Ihre Pipeline Ihre App direkt von der Quellphase aus testet, wählen Sie. MyApp Wenn Ihre Pipeline eine Build-Phase enthält, wählen Sie MyAppBuild.



15. Wählen Sie unten Add Action (Aktion hinzufügen) aus.
16. Wählen Sie im CodePipeline Bereich Pipeline-Änderung speichern und dann Änderung speichern aus.
17. Um Ihre Änderungen zu übertragen und einen Pipeline-Build zu starten, wählen Sie Release change (Änderung freigeben) und dann Release (Freigeben).

AWS CLI Referenz für AWS Device Farm

Informationen zur Verwendung von AWS Command Line Interface (AWS CLI) zum Ausführen von Device Farm-Befehlen finden Sie in der AWS CLI Referenz für AWS Device Farm.

Allgemeine Informationen zu finden Sie im AWS CLIAWS Command Line Interface Benutzerhandbuch und in der AWS CLI Befehlsreferenz.

PowerShell Windows-Referenz für AWS Device Farm

Informationen PowerShell zum Ausführen von Device Farm-Befehlen mithilfe von Windows finden Sie unter Device Farm Cmdlet-Referenz in der AWS Tools for Windows PowerShell Cmdlet-Referenz. Weitere Informationen finden Sie PowerShell im AWS -Tools für PowerShell Benutzerhandbuch unter Einrichten der AWS-Tools für Windows.

Automatisieren der AWS-Gerätefarm

Der programmatische Zugriff auf Device Farm ist eine leistungsstarke Methode, um allgemeine Aufgaben zu automatisieren, die Sie erledigen müssen, z. B. das Planen eines Laufs oder das Herunterladen der Artefakte für einen Lauf, eine Suite oder einen Test. Das AWS SDK und die AWS CLI Bereitstellung der dafür erforderlichen Mittel.

Das AWS SDK bietet Zugriff auf alle AWS Dienste, einschließlich Device Farm, Amazon S3 und mehr. Die Ausgabe des obigen Befehls sieht in etwa folgendermaßen aus (JSON format).

- die AWS Tools und SDKs
- die AWS-Gerätefarm-API-Referenz

Beispiel: Verwenden der AWS CLI oder des SDK zum Hochladen einer App oder eines Tests auf Device Farm

Die folgenden Beispiele zeigen, wie Sie mithilfe der AWS CLI oder mithilfe des AWS SDK in verschiedenen Sprachen einen Upload auf Device Farm erstellen. Uploads sind die Kernbausteine für die Planung von Testläufen auf Device Farm und beinhalten Folgendes:

- Deine App
- Dein Test
- Ihre Testspezifikationsdatei

Uploads werden mithilfe der `CreateUpload`API erstellt. Diese API gibt eine vorsignierte S3-URL zurück, auf die Sie Ihren Upload mithilfe einer HTTP-PUT-Anfrage pushen können. Die URL läuft nach 24 Stunden ab.

AWS CLI

> Hinweis: In diesem Beispiel wird das Befehlszeilentool verwendet`curl`, um die App auf Device Farm zu übertragen.
>
> Erstellen Sie zunächst ein Projekt, falls Sie dies noch nicht getan haben.

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

Dadurch werden Ausgaben wie die folgende angezeigt:

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

Gehen Sie dann wie folgt vor, um Ihren Upload zu erstellen und ihn auf Device Farm zu übertragen. In diesem Beispiel erstellen wir einen Android-App-Upload mit einer lokalen APK-Datei. Weitere Informationen zum Upload-Typ, einschließlich Details zu den Upload-Typen für iOS-Apps, finden Sie in unserer API-Dokumentation zum Erstellen eines `Upload`.

```
$ export APP_PATH="/local/path/to/my_sample_app.apk"
$ export APP_TYPE="ANDROID_APP"
```

Zuerst erstellen wir den Upload in Device Farm:

```
$ aws devicefarm create-upload \
  --project-arn "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE" \
  --name "$(basename "$APP_PATH")" \
  --type "$APP_TYPE"
```

Dadurch werden Ausgaben wie die folgende angezeigt:

```
{
    "upload": {
        "arn": "arn:aws:devicefarm:us-
west-2:385076942068:upload:490a6350-0ba3-43e5-83f5-d2896b069a34/a120e848-c57b-4e8d-
a720-d750a0c4d936",
        "name": "my_sample_app.apk",
        "created": 1760747318.266,
        "type": "ANDROID_APP",
        "status": "INITIALIZED",
        "url": "https://prod-us-west-2-uploads.s3.dualstack.us-west-2.amazonaws.com/
arn%3Aaws%3Adevicefarm%3Aus-west-2...",
        "category": "PRIVATE"
    }
```

```
}
```

Führen Sie dann einen PUT-Aufruf mit curl durch, um die App in den S3-Bucket von Device Farm zu übertragen:

```
$ curl -T "$APP_PATH" "https://prod-us-west-2-uploads.s3.dualstack.us-
west-2.amazonaws.com/arn%3Aaws%3Adevicefarm%3Aus-west-2..."
```

Warten Sie abschließend, bis sich die App im Status „Erfolgreich“ befindet:

```
$ aws devicefarm get-upload --arn "arn:aws:devicefarm:us-
west-2:385076942068:upload:490a6350-0ba3-43e5-83f5-d2896b069a34/a120e848-c57b-4e8d-
a720-d750a0c4d936"
```

Dadurch werden Ausgaben wie die folgende angezeigt:

```
{
    "upload": {
        "arn": "arn:aws:devicefarm:us-
west-2:385076942068:upload:490a6350-0ba3-43e5-83f5-d2896b069a34/a120e848-c57b-4e8d-
a720-d750a0c4d936",
        "name": "my_sample_app.apk",
        "created": 1760747318.266,
        "type": "ANDROID_APP",
        "status": "SUCCEEDED",
        "url": "https://prod-us-west-2-uploads.s3.dualstack.us-west-2.amazonaws.com/
arn%3Aaws%3Adevicefarm%3Aus-west-2...",
        "metadata": "{\"activity_name\":
\"com.amazonaws.devicefarm.android.referenceapp.Activities.MainActivity\",
\"package_name\":\"com.amazonaws.devicefarm.android.referenceapp\",...}",
        "category": "PRIVATE"
    }
}
```

Python

Hinweis: In diesem Beispiel wird das *`requests`* Drittanbieterpaket verwendet, um die App auf Device Farm zu pushen, sowie das AWS SDK für Python*`boto3`*.

Erstellen Sie zunächst ein Projekt, falls Sie dies noch nicht getan haben.

```
import boto3
```

```
client = boto3.client("devicefarm", region_name="us-west-2")
resp = client.create_project(name="MyProjectName")

print(resp)
# Response will be something like:
# {
#     "project": {
#         "name": "MyProjectName",
#         "arn": "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE",
#         "created": 1535675814.414
#     }
# }
```

Gehen Sie dann wie folgt vor, um Ihren Upload zu erstellen und ihn auf Device Farm zu übertragen. In diesem Beispiel erstellen wir einen Android-App-Upload mit einer lokalen APK-Datei. Weitere Informationen zum Upload-Typ, einschließlich Details zu den Upload-Typen für iOS-Apps, finden Sie in unserer API-Dokumentation zum Erstellen eines `Upload`.

```
import os
import time
import datetime
import requests
from pathlib import Path
import boto3


def upload_device_farm_file():
    project_arn = "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE"
    app_path = Path("/local/path/to/my_sample_app.apk")
    file_type = "ANDROID_APP"

    if not app_path.is_file():
        raise RuntimeError(f"{app_path} is not a valid app file path")

    client = boto3.client("devicefarm", region_name="us-west-2")

    # 1) Create the upload in Device Farm
    create = client.create_upload(
        projectArn=project_arn,
        name=app_path.name,
```

```
        type=file_type,
        contentType="application/octet-stream",
    )
    upload = create["upload"]
    upload_arn = upload["arn"]
    upload_url = upload["url"]
    # This will show output such as the following:
    # { "upload": { "arn": "...", "name": "my_sample_app.apk", "type":
 "ANDROID_APP", "status": "INITIALIZED", "url": "https://..." } }

    # 2) Do an HTTP PUT command to push the file to the pre-signed S3 URL
    with app_path.open("rb") as fh:
        print(f"Uploading {app_path.name} to Device Farm...")
        put_resp = requests.put(upload_url, data=fh, headers={"Content-Type":
 "application/octet-stream"})
        put_resp.raise_for_status()

    # 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
    timeout_seconds = 30
    start = time.time()
    while True:
        get_resp = client.get_upload(arn=upload_arn)
        status = get_resp["upload"]["status"]
        msg = get_resp["upload"].get("message") or
 get_resp["upload"].get("metadata") or ""
        elapsed = datetime.timedelta(seconds=int(time.time() - start))
        print(f"[{elapsed}] status={status}{' - ' + msg if msg else ''}")

        if status == "SUCCEEDED":
            print(f"Upload complete: {upload_arn}")
            return upload_arn
        if status == "FAILED":
            raise RuntimeError(f"Device Farm failed to process upload: {msg}")

        if (time.time() - start) > timeout_seconds:
            raise RuntimeError(f"Timed out after {timeout_seconds}s waiting for
 upload to process (last status={status}).")

        time.sleep(1)

upload_device_farm_file()
```

Java

Hinweis: Dieses Beispiel verwendet das AWS SDK for Java v2 und *`HttpClient`* zum Pushen der App auf Device Farm und ist mit JDK-Versionen 11 und höher kompatibel.

Erstellen Sie zunächst ein Projekt, falls Sie dies noch nicht getan haben.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import software.amazon.awssdk.services.devicefarm.model.CreateProjectRequest;
import software.amazon.awssdk.services.devicefarm.model.CreateProjectResponse;

try (DeviceFarmClient client = DeviceFarmClient.builder()
        .region(Region.US_WEST_2)
        .build()) {
    CreateProjectResponse resp = client.createProject(
        CreateProjectRequest.builder().name("MyProjectName").build());
    System.out.println(resp.project());
    // Response will be something like:
    // Project{name=MyProjectName, arn=arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-..., created=...}
}
```

Gehen Sie dann wie folgt vor, um Ihren Upload zu erstellen und ihn auf Device Farm zu übertragen. In diesem Beispiel erstellen wir einen Android-App-Upload mit einer lokalen APK-Datei. Weitere Informationen zum Upload-Typ, einschließlich Details zu den Upload-Typen für iOS-Apps, finden Sie in unserer API-Dokumentation zum Erstellen eines `Upload`.

```
import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.time.Duration;
import java.time.Instant;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import software.amazon.awssdk.services.devicefarm.model.CreateUploadRequest;
import software.amazon.awssdk.services.devicefarm.model.CreateUploadResponse;
```

```
import software.amazon.awssdk.services.devicefarm.model.GetUploadRequest;
import software.amazon.awssdk.services.devicefarm.model.GetUploadResponse;
import software.amazon.awssdk.services.devicefarm.model.Upload;
import software.amazon.awssdk.services.devicefarm.model.UploadType;

public class DeviceFarmUploader {

    public static String upload(String projectArn, Path appPath) throws Exception {
        if (projectArn == null || projectArn.isEmpty()) {
            throw new IllegalArgumentException("Missing projectArn");
        }
        if (!Files.isRegularFile(appPath)) {
            throw new IllegalArgumentException("Invalid app path: " + appPath);
        }

        String fileName = appPath.getFileName().toString().trim();
        UploadType type = UploadType.ANDROID_APP;

        // Build a reusable HttpClient
        HttpClient http = HttpClient.newBuilder()
                .version(HttpClient.Version.HTTP_1_1)
                .connectTimeout(Duration.ofSeconds(10))
                .build();

        try (DeviceFarmClient client = DeviceFarmClient.builder()
                .region(Region.US_WEST_2)
                .build()) {

            // 1) Create the upload in Device Farm
            CreateUploadResponse create =
 client.createUpload(CreateUploadRequest.builder()
                    .projectArn(projectArn)
                    .name(fileName)
                    .type(type)
                    .contentType("application/octet-stream")
                    .build());

            Upload upload = create.upload();
            String uploadArn = upload.arn();
            String url = upload.url();
            // This will show output such as the following:
            // { "upload": { "arn": "...", "name": "my_sample_app.apk", "type":
 "ANDROID_APP", "status": "INITIALIZED", "url": "https://..." } }
```

```
            // 2) PUT file to pre-signed URL using HttpClient
            HttpRequest put = HttpRequest.newBuilder(URI.create(url))
                    .timeout(Duration.ofMinutes(15))
                    .header("Content-Type", "application/octet-stream")
                    .PUT(HttpRequest.BodyPublishers.ofFile(appPath))
                    .build();

            HttpResponse<Void> resp = http.send(put,
 HttpResponse.BodyHandlers.discarding());
            int code = resp.statusCode();
            if (code / 100 != 2) {
                throw new IOException("Failed PUT to S3 pre-signed URL, HTTP " +
 code);
            }

            // 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
            Instant deadline = Instant.now().plusSeconds(30); // 30-second timeout
            while (true) {
                GetUploadResponse got = client.getUpload(GetUploadRequest.builder()
                        .arn(uploadArn)
                        .build());

                String status = got.upload().statusAsString();
                String msg = got.upload().metadata();
                System.out.println("status=" + status + (msg != null ? " - " + msg :
 ""));

                if ("SUCCEEDED".equals(status)) return uploadArn;
                if ("FAILED".equals(status)) throw new RuntimeException("Upload
 failed: " + msg);
                if (Instant.now().isAfter(deadline)) {
                    throw new RuntimeException("Timeout waiting for processing, last
 status=" + status);
                }
                Thread.sleep(2000);
            }
        }
    }

    public static void main(String[] args) throws Exception {
        String projectArn = "arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-c861-4c0a-b1d5-12345EXAMPLE";
        Path appPath = Paths.get("/local/path/to/my_sample_app.apk");
        String result = upload(projectArn, appPath);
```

```
        System.out.println("Upload ARN: " + result);
    }
}
```

JavaScript

Hinweis: In diesem Beispiel wird AWS SDK für JavaScript (v3) und Node 18+ verwendet*fetch*, um die App auf Device Farm zu pushen.

Erstellen Sie zunächst ein Projekt, falls Sie dies noch nicht getan haben.

```
import { DeviceFarmClient, CreateProjectCommand } from "@aws-sdk/client-device-
farm";

const df = new DeviceFarmClient({ region: "us-west-2" });
const resp = await df.send(new CreateProjectCommand({ name: "MyProjectName" }));
console.log(resp);
// Response will be something like:
// { project: { name: 'MyProjectName', arn: 'arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-...', created: 1535675814.414 } }
```

Gehen Sie dann wie folgt vor, um Ihren Upload zu erstellen und ihn auf Device Farm zu übertragen. In diesem Beispiel erstellen wir einen Android-App-Upload mit einer lokalen APK-Datei. Weitere Informationen zum Upload-Typ, einschließlich Details zu den Upload-Typen für iOS-Apps, finden Sie in unserer API-Dokumentation zum Erstellen eines `Upload`.

```
import { DeviceFarmClient, CreateUploadCommand, GetUploadCommand } from "@aws-sdk/
client-device-farm";
import { createReadStream } from "fs";
import { basename } from "path";

const projectArn = "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE";
const appPath = "/local/path/to/my_sample_app.apk";
const name = basename(appPath).trim();
const type = "ANDROID_APP";

const client = new DeviceFarmClient({ region: "us-west-2" });

// 1) Create the upload in Device Farm
const create = await client.send(new CreateUploadCommand({
  projectArn,
  name,
```

```
  type,
  contentType: "application/octet-stream",
}));

const uploadArn = create.upload.arn;
const url = create.upload.url;
// This will show output such as the following:
// { upload: { arn: '...', name: 'my_sample_app.apk', type: 'ANDROID_APP', status:
 'INITIALIZED', url: 'https://...' } }

// 2) PUT to pre-signed URL
const putResp = await fetch(url, {
  method: "PUT",
  headers: { "Content-Type": "application/octet-stream" },
  body: createReadStream(appPath),
});
if (!putResp.ok) {
  throw new Error(`Failed PUT to pre-signed URL: ${putResp.status} ${await
 putResp.text().catch(()=>"")}`);
}

// 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
const deadline = Date.now() + (30 * 1000); // 30-second timeout
while (true) {
  const response = await client.send(new GetUploadCommand({ arn: uploadArn }));
  const { status, message, metadata } = response.upload;
  console.log(`status=${status}${message ? " - " + message : metadata ? " - " +
 metadata : ""}`);
  if (status === "SUCCEEDED") {
    console.log("Upload complete:", uploadArn);
    break;
  }
  if (status === "FAILED") {
    throw new Error(`Upload failed: ${message || metadata || "unknown"}`);
  }
  if (Date.now() > deadline) throw new Error(`Timeout waiting for processing (last
 status=${status})`);
  await new Promise(r => setTimeout(r, 2000));
}
```

C#

Hinweis: Dieses Beispiel verwendet das AWS SDK for .NET und *`HttpClient`* um die App auf Device Farm zu pushen.

Erstellen Sie zunächst ein Projekt, falls Sie dies noch nicht getan haben.

```
using System;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

using var client = new AmazonDeviceFarmClient(RegionEndpoint.USWest2);
var resp = await client.CreateProjectAsync(new CreateProjectRequest { Name =
 "MyProjectName" });
Console.WriteLine(resp.Project);
// Response will be something like:
// { Name = MyProjectName, Arn = arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-..., Created = ... }
```

Gehen Sie dann wie folgt vor, um Ihren Upload zu erstellen und ihn auf Device Farm zu übertragen. In diesem Beispiel erstellen wir einen Android-App-Upload mit einer lokalen APK-Datei. Weitere Informationen zum Upload-Typ, einschließlich Details zu den Upload-Typen für iOS-Apps, finden Sie in unserer API-Dokumentation zum Erstellen eines `Upload`.

```
using System;
using System.IO;
using System.Net.Http;
using System.Threading.Tasks;
using System.Net.Http.Headers;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

class DeviceFarmUploader
{
    public static async Task<string> UploadAsync(string projectArn, string appPath)
    {
        if (string.IsNullOrWhiteSpace(projectArn)) throw new
 ArgumentException("Missing projectArn");
        if (!File.Exists(appPath)) throw new ArgumentException($"Invalid app path:
 {appPath}");
        var type = UploadType.ANDROID_APP;

        using var client = new AmazonDeviceFarmClient(RegionEndpoint.USWest2);
        // 1) Create the upload in Device Farm
        var create = await client.CreateUploadAsync(new CreateUploadRequest
```

```
        {
            ProjectArn = projectArn,
            Name = Path.GetFileName(appPath),
            Type = type,
            ContentType = "application/octet-stream"
        });

        var uploadArn = create.Upload.Arn;
        var url = create.Upload.Url;
        // This will show output such as the following:
        // { Upload: { Arn = ..., Name = my_sample_app.apk, Type = ANDROID_APP,
Status = INITIALIZED, Url = https://... } }

        // 2) PUT file to pre-signed URL
        using (var http = new HttpClient())
        using (var fs = File.OpenRead(appPath))
        using (var content = new StreamContent(fs))
        {
            content.Headers.Add("Content-Type", "application/octet-stream");
            var resp = await http.PutAsync(url, content);
            if (!resp.IsSuccessStatusCode)
                throw new Exception($"Failed PUT to pre-signed URL:
{(int)resp.StatusCode} {await resp.Content.ReadAsStringAsync()}");
        }

        // 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
        var deadline = DateTime.UtcNow.AddSeconds(30); // 30-second timeout
        while (true)
        {
            var got = await client.GetUploadAsync(new GetUploadRequest { Arn =
uploadArn });
            var status = got.Upload.Status.Value;
            var msg = got.Upload.Message ?? got.Upload.Metadata;
            Console.WriteLine($"status={status}{(string.IsNullOrEmpty(msg) ? "" : "
- " + msg)}");

            if (status == UploadStatus.SUCCEEDED.Value) return uploadArn;
            if (status == UploadStatus.FAILED.Value) throw new Exception($"Upload
failed: {msg}");
            if (DateTime.UtcNow > deadline) throw new TimeoutException($"Timeout
waiting for processing (last status={status})");
            await Task.Delay(2000);
        }
    }
```

```
    static async Task Main()
    {
        var projectArn = "arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-c861-4c0a-b1d5-12345EXAMPLE";
        var appPath = "/local/path/to/my_sample_app.apk";
        var result = await UploadAsync(projectArn!, appPath!);
        Console.WriteLine("Upload ARN: " + result);
    }
}
```

Ruby

Hinweis: Dieses Beispiel verwendet das AWS SDK for Ruby und *`Net::HTTP`* um die App auf Device Farm zu pushen.

Erstellen Sie zunächst ein Projekt, falls Sie dies noch nicht getan haben.

```
require "aws-sdk-devicefarm"

client = Aws::DeviceFarm::Client.new(region: "us-west-2")
resp = client.create_project(name: "MyProjectName")
puts resp.project.inspect
# Response will be something like:
# #<struct Aws::DeviceFarm::Types::Project name="MyProjectName",
 arn="arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-...",
 created=1535675814.414>
```

Gehen Sie dann wie folgt vor, um Ihren Upload zu erstellen und ihn auf Device Farm zu übertragen. In diesem Beispiel erstellen wir einen Android-App-Upload mit einer lokalen APK-Datei. Weitere Informationen zum Upload-Typ, einschließlich Details zu den Upload-Typen für iOS-Apps, finden Sie in unserer API-Dokumentation zum Erstellen eines `Upload`.

```
require "aws-sdk-devicefarm"
require "net/http"
require "uri"

project_arn = "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-c861-4c0a-
b1d5-12345EXAMPLE"
app_path    = "/local/path/to/my_sample_app.apk"
raise "Invalid APP_PATH: #{app_path}" unless File.file?(app_path)
type = "ANDROID_APP"
```

```
client = Aws::DeviceFarm::Client.new(region: "us-west-2")

# 1) Create the upload in Device Farm
create = client.create_upload(
  project_arn: project_arn,
  name: File.basename(app_path),
  type: type,
  content_type: "application/octet-stream"
)

upload_arn = create.upload.arn
url = create.upload.url
# This will show output such as the following:
# #<Upload arn="...", name="my_sample_app.apk", type="ANDROID_APP",
 status="INITIALIZED", url="https://...">

# 2) PUT the file to the pre-signed URL
uri = URI.parse(url)
Net::HTTP.start(uri.host, uri.port, use_ssl: (uri.scheme == "https")) do |http|
  req = Net::HTTP::Put.new(uri)
  req["Content-Type"] = "application/octet-stream"
  req.body_stream = File.open(app_path, "rb")
  req.content_length = File.size(app_path)
  resp = http.request(req)
  raise "Failed PUT: #{resp.code} #{resp.body}" unless resp.code.to_i / 100 == 2
end

# 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
deadline = Time.now + 30 # 30-second timeout
loop do
  got = client.get_upload(arn: upload_arn)
  status = got.upload.status
  msg = got.upload.message || got.upload.metadata
  puts "status=#{status}#{msg ? " - #{msg}" : ""}"

  case status
  when "SUCCEEDED" then puts "Upload complete: #{upload_arn}"; break
  when "FAILED"    then raise "Upload failed: #{msg}"
  end
  raise "Timeout waiting for processing (last status=#{status})" if Time.now >
 deadline
  sleep 2
end
```

Beispiel: Verwenden des AWS SDK zum Starten einer Device Farm, zum Ausführen und Sammeln von Artefakten

Das folgende Beispiel zeigt, wie Sie das AWS SDK für die Arbeit mit Device Farm verwenden können. beginning-to-end Dieses Beispiel erledigt Folgendes:

- Lädt Test- und Anwendungspakete auf Device Farm hoch
- Startet einen Testlauf und wartet auf seinen Abschluss (oder einen Fehler)
- Lädt alle von den Testsuiten produzierten Artefakte herunter

Dieses Beispiel hängt für die Interaktion mit HTTP vom `requests`-Drittanbieterpaket ab.

```
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")
```

Behebung von Gerätefarm-Fehlern

In diesem Abschnitt finden Sie Fehlermeldungen und Verfahren, mit denen Sie häufig auftretende Probleme mit Device Farm beheben können.

> **Note**
>
> Informationen zur Fehlerbehebung bei Appium-Tests, die auf Device Farm unerwartet fehlschlagen, finden Sie in unserem Leitfaden für clientseitige Appium-Tests

Themen

Fehlerbehebung bei Android-Anwendungstests in AWS Device Farm

Im folgenden Thema werden Fehlermeldungen aufgeführt, die beim Hochladen von Android-Anwendungstests auftreten können, und Umgehungen für die einzelnen Fehler empfohlen.

> **Note**
>
> Die folgenden Anweisungen gelten für Linux x86_64 and Mac.

ANDROID_APP_UNZIP_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Ihre Anwendung konnte nicht geöffnet werden. Prüfen Sie, ob die Datei gültig ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Anwendungspaket fehlerfrei dekomprimieren können. Im folgenden Beispiel ist der Name des Pakets app-debug.apk.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip app-debug.apk
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Bei einem gültigen Android-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

   ```
   .
   |-- AndroidManifest.xml
   |-- classes.dex
   |-- resources.arsc
   |-- assets (directory)
   |-- res (directory)
   `-- META-INF (directory)
   ```

   Weitere Informationen finden Sie unter Android-Tests in der AWS Device Farm.

ANDROID_APP_AAPT_DEBUG_BADGING_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir konnten keine Informationen zu Ihrer Anwendung extrahieren. Überprüfen Sie mit dem Befehl `aapt debug badging` *`<path to your test package>`*, ob die Anwendung gültig ist, und versuchen Sie es erneut, wenn der Befehl keine Fehler zurückgibt.

Während des Upload-Validierungsprozesses analysiert AWS Device Farm Informationen aus der Ausgabe eines `aapt debug badging` *`<path to your package>`* Befehls.

Stellen Sie sicher, dass Sie diesen Befehl erfolgreich für Ihre Android-Anwendung ausführen können. Im folgenden Beispiel ist der Name des Pakets app-debug.apk.

- Kopieren Sie Ihr Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den Befehl aus:

  ```
  $ aapt debug badging app-debug.apk
  ```

  Bei einem gültigen Android-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

  ```
  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'
```

Weitere Informationen finden Sie unter Android-Tests in der AWS Device Farm.

ANDROID_APP_PACKAGE_NAME_VALUE_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Der Paketname konnte in Ihrer Anwendung nicht gefunden werden. Bitte stellen Sie sicher, dass die Anwendung gültig ist, indem Sie den Befehl `aapt debug badging` *`<path to your test package>`* ausführen, und versuchen Sie es erneut, nachdem der Paketname hinter dem Schlüsselwort "„Paket: Name“ angezeigt wurde.

Während des Upload-Validierungsprozesses analysiert AWS Device Farm den Wert des Paketnamens aus der Ausgabe eines `aapt debug badging` *`<path to your package>`* Befehls.

Stellen Sie sicher, dass Sie diesen Befehl für Ihre Android-Anwendung ausführen und den Wert für den Paketnamen erfolgreich finden können. Im folgenden Beispiel ist der Name des Pakets app-debug.apk.

- Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

  ```
  $ aapt debug badging app-debug.apk | grep "package: name="
  ```

  Bei einem gültigen Android-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

  ```
  package: name='com.amazon.aws.adf.android.referenceapp' versionCode='1'
   versionName='1.0' platformBuildVersionName='5.1.1-1819727'
  ```

  Weitere Informationen finden Sie unter Android-Tests in der AWS Device Farm.

ANDROID_APP_SDK_VERSION_VALUE_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Die SDK-Version konnte in Ihrer Anwendung nicht gefunden werden. Bitte stellen Sie sicher, dass die Anwendung gültig ist, indem Sie den Befehl `aapt debug badging` *`<path to your test package>`* ausführen, und versuchen Sie es erneut, nachdem die SDK-Version hinter dem Schlüsselwort `sdkVersion` angezeigt wurde.

Während des Upload-Validierungsprozesses analysiert AWS Device Farm den SDK-Versionswert aus der Ausgabe eines `aapt debug badging` *`<path to your package>`* Befehls.

Stellen Sie sicher, dass Sie diesen Befehl für Ihre Android-Anwendung ausführen und den Wert für den Paketnamen erfolgreich finden können. Im folgenden Beispiel ist der Name des Pakets app-debug.apk.

- Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

  ```
  $ aapt debug badging app-debug.apk | grep "sdkVersion"
  ```

  Bei einem gültigen Android-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

  ```
  sdkVersion:'9'
  ```

  Weitere Informationen finden Sie unter Android-Tests in der AWS Device Farm.

ANDROID_APP_AAPT_DUMP_XMLTREE_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir konnten die gültige AndroidManifest XML-Datei in Ihrer Anwendung nicht finden. Überprüfen Sie mit dem Befehl `aapt dump xmltree` *`<path to your test package>`* `AndroidManifest.xml`, ob das Testpaket gültig ist, und versuchen Sie es erneut, wenn der Befehl keine Fehler mehr zurückgibt.

Während des Upload-Validierungsprozesses analysiert AWS Device Farm mithilfe des Befehls Informationen aus dem XML-Analysebaum nach einer im Paket enthaltenen XML-Datei. `aapt dump xmltree` *`<path to your package>`* `AndroidManifest.xml`

Stellen Sie sicher, dass Sie diesen Befehl erfolgreich für Ihre Android-Anwendung ausführen können. Im folgenden Beispiel ist der Name des Pakets app-debug.apk.

- Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

  ```
  $ aapt dump xmltree app-debug.apk. AndroidManifest.xml
  ```

  Bei einem gültigen Android-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

  ```
  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")
  ```

Weitere Informationen finden Sie unter Android-Tests in der AWS Device Farm.

ANDROID_APP_DEVICE_ADMIN_PERMISSIONS

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir haben festgestellt, dass Ihre Anwendung Administratorberechtigungen für das Gerät erfordert. Bitte stellen Sie sicher, dass die Berechtigungen nicht erforderlich sind, indem Sie den Befehl `aapt dump xmltree <path to your test package> AndroidManifest.xml` ausführen, und versuchen Sie es erneut, nachdem Sie sichergestellt haben, dass die Ausgabe das Stichwort `android.permission.BIND_DEVICE_ADMIN` nicht enthält.

Während des Upload-Validierungsprozesses analysiert AWS Device Farm mithilfe des Befehls die Berechtigungsinformationen aus dem XML-Analysebaum für eine im Paket enthaltene XML-Datei. `aapt dump xmltree <path to your package> AndroidManifest.xml`

Stellen Sie sicher, dass Ihre Anwendung keine Administratorberechtigung für das Gerät benötigt. Im folgenden Beispiel ist der Name des Pakets app-debug.apk.

- Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

  ```
  $ aapt dump xmltree app-debug.apk AndroidManifest.xml
  ```

  Die Ausgabe sollte folgendermaßen aussehen:

  ```
  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")
        ……
```

Wenn die Android-Anwendung gültig ist, sollte die Ausgabe Folgendes nicht enthalten: `A: android:name(0x01010003)="android.permission.BIND_DEVICE_ADMIN" (Raw: "android.permission.BIND_DEVICE_ADMIN")`.

Weitere Informationen finden Sie unter Android-Tests in der AWS Device Farm.

Bestimmte Fenster in meiner Android-Anwendung zeigen einen leeren oder schwarzen Bildschirm

Wenn Sie eine Android-Anwendung testen und feststellen, dass bestimmte Fenster in der Anwendung in der Videoaufzeichnung Ihres Tests von Device Farm mit einem schwarzen Bildschirm angezeigt werden, verwendet Ihre Anwendung möglicherweise die `FLAG_SECURE` Android-Funktion. Diese Markierung (wie in der offiziellen Dokumentation von Android beschrieben) wird verwendet, um zu verhindern, dass bestimmte Fenster einer Anwendung von Bildschirmaufzeichnungstools aufgezeichnet werden. Daher zeigt die Bildschirmaufzeichnungsfunktion von Device Farm (sowohl für Automatisierungs- als auch für Fernzugriffstests) möglicherweise einen schwarzen Bildschirm anstelle des Fensters Ihrer Anwendung an, wenn das Fenster diese Flagge verwendet.

Dieses Kennzeichen wird häufig von Entwicklern für Seiten in ihrer Anwendung verwendet, die vertrauliche Informationen enthalten, z. B. Anmeldeseiten. Wenn Sie für bestimmte Seiten wie die Anmeldeseite Ihrer Anwendung einen schwarzen Bildschirm anstelle des Bildschirms Ihrer Anwendung sehen, arbeiten Sie mit Ihren Entwicklern zusammen, um einen Build der Anwendung zu erhalten, der dieses Kennzeichen nicht zum Testen verwendet.

Beachten Sie außerdem, dass Device Farm weiterhin mit Anwendungsfenstern interagieren kann, die dieses Flag haben. Wenn also die Anmeldeseite Ihrer Anwendung als schwarzer Bildschirm angezeigt wird, können Sie möglicherweise immer noch Ihre Anmeldeinformationen eingeben, um

sich bei der Anwendung anzumelden (und so Seiten anzuzeigen, die nicht durch die `FLAG_SECURE` Markierung blockiert wurden).

Fehlerbehebung bei JUnit Appium-Java-Tests in AWS Device Farm

Im folgenden Thema sind Fehlermeldungen aufgeführt, die beim Hochladen von JUnit Appium-Java-Tests auftreten, und es werden Lösungsansätze zur Behebung der einzelnen Fehler empfohlen.

> **Note**
>
> Die folgenden Anweisungen gelten für Linux x86_64 and Mac.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Ihre ZIP-Datei für den Test konnte nicht geöffnet werden. Prüfen Sie, ob die Datei gültig ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets .zip. zip-with-dependencies

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Ein gültiges JUnit Appium-Java-Paket sollte eine Ausgabe wie die folgende erzeugen:

```
.
|– 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
```

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_DEPENDENCY_DIR_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Das Verzeichnis dependency-jars konnte in Ihrem Testpaket nicht gefunden werden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob das Verzeichnis dependency-jars in dem Paket enthalten ist, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets zip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das JUnit Appium-Java-Paket gültig ist, finden Sie das *dependency-jars* Verzeichnis im Arbeitsverzeichnis:

```
.
|— 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
```

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DI

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> In der Verzeichnisstruktur von „dependency-jars“ konnte keine JAR-Datei gefunden werden. Extrahieren Sie Ihr Testpaket und öffnen Sie dann das Verzeichnis „dependency-jars”; überprüfen Sie, ob sich im Verzeichnis mindestens eine JAR-Datei befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets zip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das JUnit Appium-Java-Paket gültig ist, finden Sie mindestens eine *jar* Datei im *dependency-jars* Verzeichnis:

```
.
|— 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
```

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> In Ihrem Testpaket konnte keine „*-tests.jar“-Datei gefunden werden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob mindestens eine „*-tests.jar” in dem Paket enthalten ist, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets zip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Wenn das JUnit Appium-Java-Paket gültig ist, finden Sie mindestens eine *jar* Datei wie *acme-android-appium-1.0-SNAPSHOT-tests.jar* in unserem Beispiel. Der Name der Datei kann unterschiedlich sein, sollte aber mit *-tests.jar* enden.

   ```
   .
   |— 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
   ```

   Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JA

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> In der JAR-Datei der Tests konnte keine Klassendatei gefunden werden. Extrahieren Sie Ihr ZIP-Testpaket und dekomprimieren Sie dann die JAR-Datei für die Tests; überprüfen Sie, ob sich in der JAR-Datei mindestens eine Klassendatei befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets zip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Sie sollten mindestens eine JAR-Datei wie *acme-android-appium-1.0-SNAPSHOT-tests.jar* in unserem Beispiel finden. Der Name der Datei kann unterschiedlich sein, sollte aber mit enden*-tests.jar*.

   ```
   .
   |— 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. Nachdem Sie die Dateien erfolgreich extrahiert haben, sollte in der Struktur des Arbeitsverzeichnisses bei der Ausführung des folgenden Befehls mindestens eine Klasse angezeigt werden:

```
$ tree .
```

Sie sollten eine Ausgabe wie die Folgende sehen:

```
.
|— 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
```

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_JUNIT_VERSION_VALUE_UNKNOWN

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir konnten keinen JUnit Versionswert finden. Bitte entpacken Sie Ihr Testpaket und öffnen Sie das Verzeichnis dependency-jars, überprüfen Sie, ob sich die JUnit JAR-Datei im Verzeichnis befindet, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets .zip. zip-with-dependencies

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
tree .
```

Die Ausgabe sollte in etwa wie folgt aussehen:

```
.
|— 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
```

Wenn das JUnit Appium-Java-Paket gültig ist, finden Sie die JUnit Abhängigkeitsdatei, die der JAR-Datei *junit-4.10.jar* in unserem Beispiel ähnelt. Der Name sollte aus dem Schlüsselwort *junit* und seiner Versionsnummer bestehen, die in diesem Beispiel 4.10 lautet.

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir haben festgestellt, dass die JUnit Version niedriger als die von uns unterstützte Mindestversion 4.10 war. Bitte ändern Sie die JUnit Version und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets zip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Sie sollten eine JUnit Abhängigkeitsdatei wie *junit-4.10.jar* in unserem Beispiel und ihre Versionsnummer finden, die in unserem Beispiel 4.10 lautet:

   ```
   .
   |— 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**
   >
   > Ihre Tests werden möglicherweise nicht korrekt ausgeführt, wenn die in Ihrem Testpaket angegebene JUnit Version niedriger ist als die von uns unterstützte Mindestversion 4.10.

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

Fehlerbehebung bei Appium JUnit Java-Webanwendungstests in AWS Device Farm

Im folgenden Thema sind Fehlermeldungen aufgeführt, die beim Hochladen von Appium JUnit Java-Webanwendungstests auftreten, und es werden Lösungsansätze zur Behebung der einzelnen Fehler empfohlen. Weitere Informationen zur Verwendung von Appium mit Device Farm finden Sie unterthe section called “Automatische Appium-Tests”.

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Ihre ZIP-Datei für den Test konnte nicht geöffnet werden. Prüfen Sie, ob die Datei gültig ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets zip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Ein gültiges JUnit Appium-Java-Paket sollte eine Ausgabe wie die folgende erzeugen:

```
.
|– 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

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Das Verzeichnis dependency-jars konnte in Ihrem Testpaket nicht gefunden werden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob das Verzeichnis dependency-jars in dem Paket enthalten ist, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets zip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

Wenn das JUnit Appium-Java-Paket gültig ist, finden Sie das *dependency-jars* Verzeichnis im Arbeitsverzeichnis:

```
.
|– 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

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> In der Verzeichnisstruktur von „dependency-jars“ konnte keine JAR-Datei gefunden werden. Extrahieren Sie Ihr Testpaket und öffnen Sie dann das Verzeichnis „dependency-jars”; überprüfen Sie, ob sich im Verzeichnis mindestens eine JAR-Datei befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets zip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das JUnit Appium-Java-Paket gültig ist, finden Sie mindestens eine *jar* Datei im *dependency-jars* Verzeichnis:

```
.
|– 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

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> In Ihrem Testpaket konnte keine „*-tests.jar“-Datei gefunden werden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob mindestens eine „*-tests.jar” in dem Paket enthalten ist, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets zip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das JUnit Appium-Java-Paket gültig ist, finden Sie mindestens eine *jar* Datei wie *acme-android-appium-1.0-SNAPSHOT-tests.jar* in unserem Beispiel. Der Name der Datei kann unterschiedlich sein, sollte aber mit *-tests.jar* enden.

```
.
|— 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

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> In der JAR-Datei der Tests konnte keine Klassendatei gefunden werden. Extrahieren Sie Ihr ZIP-Testpaket und dekomprimieren Sie dann die JAR-Datei für die Tests; überprüfen Sie, ob sich in der JAR-Datei mindestens eine Klassendatei befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets zip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Sie sollten mindestens eine JAR-Datei wie *acme-android-appium-1.0-SNAPSHOT-tests.jar* in unserem Beispiel finden. Der Name der Datei kann unterschiedlich sein, sollte aber mit enden*-tests.jar*.

```
.
|— 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. Nachdem Sie die Dateien erfolgreich extrahiert haben, sollte in der Struktur des Arbeitsverzeichnisses bei der Ausführung des folgenden Befehls mindestens eine Klasse angezeigt werden:

```
$ tree .
```

Sie sollten eine Ausgabe wie die Folgende sehen:

```
.
|— 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

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir konnten keinen JUnit Versionswert finden. Bitte entpacken Sie Ihr Testpaket und öffnen Sie das Verzeichnis dependency-jars, überprüfen Sie, ob sich die JUnit JAR-Datei im Verzeichnis befindet, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets .zip. zip-with-dependencies

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   tree .
   ```

   Die Ausgabe sollte in etwa wie folgt aussehen:

   ```
   .
   |– 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
```

Wenn das JUnit Appium-Java-Paket gültig ist, finden Sie die JUnit Abhängigkeitsdatei, die der JAR-Datei *junit-4.10.jar* in unserem Beispiel ähnelt. Der Name sollte aus dem Schlüsselwort *junit* und seiner Versionsnummer bestehen, die in diesem Beispiel 4.10 lautet.

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir haben festgestellt, dass die JUnit Version niedriger als die von uns unterstützte Mindestversion 4.10 war. Bitte ändern Sie die JUnit Version und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets zip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Sie sollten eine JUnit Abhängigkeitsdatei wie *junit-4.10.jar* in unserem Beispiel und ihre Versionsnummer finden, die in unserem Beispiel 4.10 lautet:

```
.
|– 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**
>
> Ihre Tests werden möglicherweise nicht korrekt ausgeführt, wenn die in Ihrem Testpaket angegebene JUnit Version niedriger ist als die von uns unterstützte Mindestversion 4.10.

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

Fehlerbehebung bei Appium Java TestNG-Tests in AWS Device Farm

Im folgenden Thema werden Fehlermeldungen aufgeführt, die beim Hochladen von Appium Java TestNG-Tests auftreten können, und Umgehungen für die einzelnen Fehler empfohlen.

> **Note**
>
> Die folgenden Anweisungen gelten für Linux x86_64 and Mac.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Ihre ZIP-Datei für den Test konnte nicht geöffnet werden. Prüfen Sie, ob die Datei gültig ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets zip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Ein gültiges JUnit Appium-Java-Paket sollte eine Ausgabe wie die folgende erzeugen:

   ```
   .
   |— 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
   ```

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Das Verzeichnis `dependency-jars` konnte in Ihrem Testpaket nicht gefunden werden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob das Verzeichnis `dependency-jars` im Paket enthalten ist, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets zip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Wenn das JUnit Appium-Java-Paket gültig ist, finden Sie das *`dependency-jars`* Verzeichnis im Arbeitsverzeichnis.

   ```
   .
   |— 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
```

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> In der Verzeichnisstruktur von „dependency-jars“ konnte keine JAR-Datei gefunden werden. Extrahieren Sie Ihr Testpaket und öffnen Sie dann das Verzeichnis „dependency-jars”; überprüfen Sie, ob sich im Verzeichnis mindestens eine JAR-Datei befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets zip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Wenn das JUnit Appium-Java-Paket gültig ist, finden Sie mindestens eine *jar* Datei im *dependency-jars* Verzeichnis.

   ```
   .
   |— 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
```

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> In Ihrem Testpaket konnte keine „*-tests.jar“-Datei gefunden werden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob mindestens eine „*-tests.jar” in dem Paket enthalten ist, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets zip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

Wenn das JUnit Appium-Java-Paket gültig ist, finden Sie mindestens eine *jar* Datei wie *acme-android-appium-1.0-SNAPSHOT-tests.jar* in unserem Beispiel. Der Name der Datei kann unterschiedlich sein, sollte aber mit *-tests.jar* enden.

```
.
|– 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
```

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> In der JAR-Datei der Tests konnte keine Klassendatei gefunden werden. Extrahieren Sie Ihr ZIP-Testpaket und dekomprimieren Sie dann die JAR-Datei für die Tests; überprüfen Sie, ob sich in der JAR-Datei mindestens eine Klassendatei befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets zip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Sie sollten mindestens eine JAR-Datei wie *acme-android-appium-1.0-SNAPSHOT-tests.jar* in unserem Beispiel finden. Der Name der Datei kann unterschiedlich sein, sollte aber mit enden*-tests.jar*.

```
.
|— 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. Um Dateien aus der jar-Datei zu extrahieren, können Sie den folgenden Befehl ausführen:

```
$ jar xf acme-android-appium-1.0-SNAPSHOT-tests.jar
```

4. Nachdem Sie die Dateien erfolgreich extrahiert haben, führen Sie den folgenden Befehl aus:

```
$ tree .
```

Sie sollten mindestens eine Klasse in der Arbeitsverzeichnisstruktur finden:

```
.
|— 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
```

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

Fehlerbehebung bei Appium Java TestNG-Webanwendungen in AWS Device Farm

Im folgenden Thema werden Fehlermeldungen aufgelistet, die beim Hochladen von Appium Java TestNG-Tests für Webanwendungen auftreten können, und Behelfslösungen für die einzelnen Fehler empfohlen.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Ihre ZIP-Datei für den Test konnte nicht geöffnet werden. Prüfen Sie, ob die Datei gültig ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets .zip. zip-with-dependencies

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Ein gültiges JUnit Appium-Java-Paket sollte eine Ausgabe wie die folgende erzeugen:

```
.
|— 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
```

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSIN

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Das Verzeichnis dependency-jars konnte in Ihrem Testpaket nicht gefunden werden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob das Verzeichnis dependency-jars in dem Paket enthalten ist, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets zip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das JUnit Appium-Java-Paket gültig ist, finden Sie das *dependency-jars* Verzeichnis im Arbeitsverzeichnis.

```
.
|— 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
```

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDE

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> In der Verzeichnisstruktur von „dependency-jars“ konnte keine JAR-Datei gefunden werden. Extrahieren Sie Ihr Testpaket und öffnen Sie dann das Verzeichnis „dependency-jars”; überprüfen Sie, ob sich im Verzeichnis mindestens eine JAR-Datei befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets zip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Wenn das JUnit Appium-Java-Paket gültig ist, finden Sie mindestens eine *jar* Datei im *dependency-jars* Verzeichnis.

   ```
   .
   |— 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
   ```

   Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> In Ihrem Testpaket konnte keine „*-tests.jar“-Datei gefunden werden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob mindestens eine „*-tests.jar” in dem Paket enthalten ist, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets zip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Wenn das JUnit Appium-Java-Paket gültig ist, finden Sie mindestens eine *jar* Datei wie *acme-android-appium-1.0-SNAPSHOT-tests.jar* in unserem Beispiel. Der Name der Datei kann unterschiedlich sein, sollte aber mit *-tests.jar* enden.

   ```
   .
   |— 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
   ```

   Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_T

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> In der JAR-Datei der Tests konnte keine Klassendatei gefunden werden. Extrahieren Sie Ihr ZIP-Testpaket und dekomprimieren Sie dann die JAR-Datei für die Tests; überprüfen Sie, ob sich in der JAR-Datei mindestens eine Klassendatei befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets zip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Sie sollten mindestens eine JAR-Datei wie *acme-android-appium-1.0-SNAPSHOT-tests.jar* in unserem Beispiel finden. Der Name der Datei kann unterschiedlich sein, sollte aber mit enden*-tests.jar*.

   ```
   .
   |— 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. Um Dateien aus der jar-Datei zu extrahieren, können Sie den folgenden Befehl ausführen:

```
$ jar xf acme-android-appium-1.0-SNAPSHOT-tests.jar
```

4. Nachdem Sie die Dateien erfolgreich extrahiert haben, führen Sie den folgenden Befehl aus:

```
$ tree .
```

Sie sollten mindestens eine Klasse in der Arbeitsverzeichnisstruktur finden:

```
.
|– 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
```

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

Fehlerbehebung bei Appium-Python-Tests in AWS Device Farm

Im folgenden Thema werden Fehlermeldungen aufgeführt, die beim Hochladen von Appium Python-Tests auftreten können, und Behelfslösungen für die einzelnen Fehler empfohlen.

APPIUM_PYTHON_TEST_PACKAGE_UNZIP_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir konnten Ihre ZIP-Datei für den Appium-Test nicht öffnen. Prüfen Sie, ob die Datei gültig ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets test_bundle.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip test_bundle.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Bei einem gültigen Appium Python-Paket sollte die Ausgabe wie folgt aussehen:

   ```
   .
   |-- 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
   ```

   Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir konnten keine Wheel-Datei für die Abhängigkeiten in der Wheelhouse-Verzeichnisstruktur finden. Extrahieren Sie Ihr Testpaket und öffnen Sie dann das Verzeichnis „wheelhouse”; überprüfen Sie, ob sich im Verzeichnis mindestens eine Wheel-Datei befindet, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets test_bundle.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip test_bundle.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Wenn das Appium-Python-Paket gültig ist, finden Sie mindestens eine *.whl* abhängige Datei wie die hervorgehobenen Dateien im *wheelhouse* Verzeichnis.

   ```
   .
   |-- 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
```

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_INVALID_PLATFORM

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir haben festgestellt, dass in mindestens einer Wheel-Datei eine Plattform angegeben ist, die nicht unterstützt wird. Extrahieren Sie Ihr Testpaket und öffnen Sie dann das Verzeichnis „wheelhouse”; überprüfen Sie, ob die Namen der Wheel-Dateien auf -any.whl oder -linux_x86_64.whl enden, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets test_bundle.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip test_bundle.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Wenn das Appium-Python-Paket gültig ist, finden Sie mindestens eine *.whl* abhängige Datei wie die hervorgehobenen Dateien im *wheelhouse* Verzeichnis. Der Name der Datei kann unterschiedlich sein, aber er sollte mit *-any.whl* oder enden*-linux_x86_64.whl*, was die Plattform angibt. Andere Plattformen wie `windows` werden nicht unterstützt.

   ```
   .
   |-- 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
```

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir konnten das Verzeichnis „tests” nicht in Ihrem Testpaket finden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob das Verzeichnis „tests” in dem Paket enthalten ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets test_bundle.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip test_bundle.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

Wenn das Appium-Python-Paket gültig ist, finden Sie das *tests* Verzeichnis im Arbeitsverzeichnis.

```
.
|-- 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
```

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir konnten keine gültige Test-Datei in der Verzeichnisstruktur für die Tests finden. Extrahieren Sie Ihr Paket und öffnen Sie dann das Verzeichnis „tests”; überprüfen Sie, ob der Name mindestens einer Datei mit dem Schlüsselwort „test” beginnt oder darauf endet, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets test_bundle.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip test_bundle.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das Appium-Python-Paket gültig ist, finden Sie das *tests* Verzeichnis im Arbeitsverzeichnis. Der Name der Datei kann unterschiedlich sein, aber er sollte mit beginnen *test_* oder enden mit*_test.py*.

```
.
|-- 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
```

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir konnten die Datei „requirements.txt” nicht in Ihrem Testpaket finden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob die Datei „requirements.txt” in dem Paket enthalten ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets test_bundle.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip test_bundle.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das Appium-Python-Paket gültig ist, finden Sie die *requirements.txt* Datei im Arbeitsverzeichnis.

```
.
|-- 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
```

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir haben festgestellt, dass pytest in einer Version niedriger als 2.8.0 verwendet wird, der minimalen unterstützten Version. Bitte ändern Sie die Version von pytest in der Datei „requirements.txt” und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets test_bundle.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip test_bundle.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Sie sollten die *requirements.txt* Datei im Arbeitsverzeichnis finden.

   ```
   .
   |-- 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. Sie können die Version von pytest abrufen, indem Sie den folgenden Befehl ausführen:

   ```
   $ grep "pytest" requirements.txt
   ```

   Die Ausgabe sollte folgendermaßen aussehen:

   ```
   pytest==2.9.0
   ```

   Die Version von pytest wird angezeigt, in diesem Beispiel 2.9.0. Wenn das Appium Python-Paket gültig ist, sollte pytest in der Version 2.8.0 oder höher verwendet werden.

   Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS_FAIL

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir konnten die abhängigen Wheel-Dateien nicht installieren. Extrahieren Sie Ihr Testpaket und öffnen Sie die Datei „requirements.txt” sowie das Verzeichnis „wheelhouse”; überprüfen Sie, ob die in der Datei „requirements.txt” angegebenen abhängigen Wheel-Dateien mit denen im Verzeichnis „wheelhouse” übereinstimmen, und versuchen Sie es erneut.

Wir empfehlen dringend, das Modul Python virtualenv für die Überprüfung von Paketen zu installieren. Das folgende Beispiel zeigt, wie Sie ein virtuelle Umgebung mit Python virtualenv erstellen und diese anschließend aktivieren:

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

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets test_bundle.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip test_bundle.zip
   ```

2. Sie können die Installation von Wheel-Dateien testen, indem Sie den folgenden Befehl ausführen:

   ```
   $ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
   requirements.txt
   ```

   Bei einem gültigen Appium Python-Paket sollte die Ausgabe wie folgt aussehen:

   ```
   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. Sie können die virtuelle Umgebung deaktivieren, indem Sie den folgenden Befehl ausführen:

```
$ deactivate
```

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir konnten im Verzeichnis „tests” keine Tests erfassen. Extrahieren Sie Ihr Testpaket, überprüfen Sie mit dem Befehl `py.test --collect-only <path to your tests directory>`, ob das Testpaket gültig ist, und versuchen Sie es erneut, wenn der Befehl keine Fehler zurückgibt.

Wir empfehlen dringend, das Modul Python virtualenv für die Überprüfung von Paketen zu installieren. Das folgende Beispiel zeigt, wie Sie ein virtuelle Umgebung mit Python virtualenv erstellen und diese anschließend aktivieren:

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

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets test_bundle.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip test_bundle.zip
   ```

2. Sie können die Wheel-Dateien installieren, indem Sie den folgenden Befehl ausführen:

   ```
   $ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
   requirements.txt
   ```

3. Sie können die Tests erfassen, indem Sie den folgenden Befehl ausführen:

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

   Bei einem gültigen Appium Python-Paket sollte die Ausgabe wie folgt aussehen:

   ```
   ==================== 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. Sie können die virtuelle Umgebung deaktivieren, indem Sie den folgenden Befehl ausführen:

   ```
   $ deactivate
   ```

   Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEELS_INSUFFICIENT

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir konnten im Wheelhouse-Verzeichnis nicht genügend Radabhängigkeiten finden. Bitte entpacken Sie Ihr Testpaket und öffnen Sie dann das Wheelhouse-Verzeichnis. Stellen Sie sicher, dass Sie alle Radabhängigkeiten in der Datei requirements.txt angegeben haben.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets test_bundle.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip test_bundle.zip
   ```

2. Überprüfen Sie die Länge der *requirements.txt* Datei sowie die Anzahl der *.whl* abhängigen Dateien im Wheelhouse-Verzeichnis:

   ```
   $ cat requirements.txt | egrep "." |wc -l
       12
   $ ls wheelhouse/ | egrep ".+\.whl" | wc -l
       11
   ```

   Wenn die Anzahl der *.whl* abhängigen Dateien geringer ist als die Anzahl der nicht leeren Zeilen in Ihrer *requirements.txt* Datei, müssen Sie Folgendes sicherstellen:

   - Jeder Zeile in der Datei entspricht eine *.whl* abhängige *requirements.txt* Datei.
   - Es gibt keine anderen Zeilen in der *requirements.txt* Datei, die andere Informationen als die Namen der Abhängigkeitspakete enthalten.
   - In der *requirements.txt* Datei werden keine Abhängigkeitsnamen in mehreren Zeilen dupliziert, sodass zwei Zeilen in der Datei einer *.whl* abhängigen Datei entsprechen können.

   AWS Device Farm unterstützt keine Zeilen in der *requirements.txt* Datei, die nicht direkt Abhängigkeitspaketen entsprechen, wie z. B. Zeilen, die globale Optionen für den `pip install` Befehl angeben. Eine Liste der globalen Optionen finden Sie unter Anforderungsdateiformat.

   Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

Fehlerbehebung bei Appium-Python-Webanwendungstests in AWS Device Farm

Im folgenden Thema werden Fehlermeldungen aufgelistet, die beim Hochladen von Appium Python for Web Application-Tests auftreten können, und Umgehungen für die einzelnen Fehler empfohlen.

APPIUM_WEB_PYTHON_TEST_PACKAGE_UNZIP_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir konnten Ihre ZIP-Datei für den Appium-Test nicht öffnen. Prüfen Sie, ob die Datei gültig ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets test_bundle.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip test_bundle.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Bei einem gültigen Appium Python-Paket sollte die Ausgabe wie folgt aussehen:

   ```
   .
   |-- 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
```

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir konnten keine Wheel-Datei für die Abhängigkeiten in der Wheelhouse-Verzeichnisstruktur finden. Extrahieren Sie Ihr Testpaket und öffnen Sie dann das Verzeichnis „wheelhouse”; überprüfen Sie, ob sich im Verzeichnis mindestens eine Wheel-Datei befindet, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets test_bundle.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip test_bundle.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Wenn das Appium-Python-Paket gültig ist, finden Sie mindestens eine *.whl* abhängige Datei wie die hervorgehobenen Dateien im *wheelhouse* Verzeichnis.

   ```
   .
   ```

```
|-- 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
```

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PLATFORM

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir haben festgestellt, dass in mindestens einer Wheel-Datei eine Plattform angegeben ist, die nicht unterstützt wird. Extrahieren Sie Ihr Testpaket und öffnen Sie dann das Verzeichnis „wheelhouse”; überprüfen Sie, ob die Namen der Wheel-Dateien auf -any.whl oder -linux_x86_64.whl enden, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets test_bundle.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip test_bundle.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

Wenn das Appium-Python-Paket gültig ist, finden Sie mindestens eine *.whl* abhängige Datei wie die hervorgehobenen Dateien im *wheelhouse* Verzeichnis. Der Name der Datei kann unterschiedlich sein, aber er sollte mit *-any.whl* oder enden*-linux_x86_64.whl*, was die Plattform angibt. Andere Plattformen wie `windows` werden nicht unterstützt.

```
.
|-- 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
```

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir konnten das Verzeichnis „tests” nicht in Ihrem Testpaket finden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob das Verzeichnis „tests” in dem Paket enthalten ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets test_bundle.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip test_bundle.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das Appium-Python-Paket gültig ist, finden Sie das *tests* Verzeichnis im Arbeitsverzeichnis.

```
.
|-- 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
```

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir konnten keine gültige Test-Datei in der Verzeichnisstruktur für die Tests finden. Extrahieren Sie Ihr Paket und öffnen Sie dann das Verzeichnis „tests”; überprüfen Sie, ob der Name mindestens einer Datei mit dem Schlüsselwort „test” beginnt oder darauf endet, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets test_bundle.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip test_bundle.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Wenn das Appium-Python-Paket gültig ist, finden Sie das *tests* Verzeichnis im Arbeitsverzeichnis. Der Name der Datei kann unterschiedlich sein, aber er sollte mit beginnen *test_* oder enden mit*_test.py*.

   ```
   .
   |-- 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
   ```

   Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISS

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir konnten die Datei „requirements.txt” nicht in Ihrem Testpaket finden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob die Datei „requirements.txt” in dem Paket enthalten ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets test_bundle.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip test_bundle.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Wenn das Appium-Python-Paket gültig ist, finden Sie die *`requirements.txt`* Datei im Arbeitsverzeichnis.

   ```
   .
   |-- 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
   ```

   Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir haben festgestellt, dass pytest in einer Version niedriger als 2.8.0 verwendet wird, der minimalen unterstützten Version. Bitte ändern Sie die Version von pytest in der Datei „requirements.txt” und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets test_bundle.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip test_bundle.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Sie sollten die *requirement.txt* Datei im Arbeitsverzeichnis finden.

   ```
   .
   |-- 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. Sie können die Version von pytest abrufen, indem Sie den folgenden Befehl ausführen:

```
$ grep "pytest" requirements.txt
```

Die Ausgabe sollte folgendermaßen aussehen:

```
pytest==2.9.0
```

Die Version von pytest wird angezeigt, in diesem Beispiel 2.9.0. Wenn das Appium Python-Paket gültig ist, sollte pytest in der Version 2.8.0 oder höher verwendet werden.

Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir konnten die abhängigen Wheel-Dateien nicht installieren. Extrahieren Sie Ihr Testpaket und öffnen Sie die Datei „requirements.txt” sowie das Verzeichnis „wheelhouse”; überprüfen Sie, ob die in der Datei „requirements.txt” angegebenen abhängigen Wheel-Dateien mit denen im Verzeichnis „wheelhouse” übereinstimmen, und versuchen Sie es erneut.

Wir empfehlen dringend, das Modul Python virtualenv für die Überprüfung von Paketen zu installieren. Das folgende Beispiel zeigt, wie Sie ein virtuelle Umgebung mit Python virtualenv erstellen und diese anschließend aktivieren:

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

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets test_bundle.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip test_bundle.zip
```

2. Sie können die Installation von Wheel-Dateien testen, indem Sie den folgenden Befehl ausführen:

```
$ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
requirements.txt
```

   Bei einem gültigen Appium Python-Paket sollte die Ausgabe wie folgt aussehen:

```
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. Sie können die virtuelle Umgebung deaktivieren, indem Sie den folgenden Befehl ausführen:

```
$ deactivate
```

   Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir konnten im Verzeichnis „tests” keine Tests erfassen. Extrahieren Sie Ihr Testpaket, überprüfen Sie mit dem Befehl "py.test --collect-only <Pfad zu Ihrem Testverzeichnis>", ob das Testpaket gültig ist, und versuchen Sie es erneut, wenn der Befehl keine Fehler zurückgibt.

Wir empfehlen dringend, das Modul Python virtualenv für die Überprüfung von Paketen zu installieren. Das folgende Beispiel zeigt, wie Sie ein virtuelle Umgebung mit Python virtualenv erstellen und diese anschließend aktivieren:

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

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets test_bundle.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip test_bundle.zip
   ```

2. Sie können die Wheel-Dateien installieren, indem Sie den folgenden Befehl ausführen:

   ```
   $ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
   requirements.txt
   ```

3. Sie können die Tests erfassen, indem Sie den folgenden Befehl ausführen:

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

   Bei einem gültigen Appium Python-Paket sollte die Ausgabe wie folgt aussehen:

   ```
   ==================== 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. Sie können die virtuelle Umgebung deaktivieren, indem Sie den folgenden Befehl ausführen:

   ```
   $ deactivate
   ```

   Weitere Informationen finden Sie unter Automatisches Ausführen von Appium-Tests in Device Farm.

Fehlerbehebung bei Instrumentierungstests in AWS Device Farm

Im folgenden Thema werden Fehlermeldungen aufgeführt, die beim Hochladen von Instrumentierungstests auftreten können, und Umgehungen für die einzelnen Fehler empfohlen.

> **Note**
>
> Wichtige Überlegungen zur Verwendung von Instrumentierungstests in AWS Device Farm finden Sie unterInstrumentierung für Android und AWS Device Farm.

INSTRUMENTATION_TEST_PACKAGE_UNZIP_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

```
Warning: We could not open your test APK file. Please verify that the file is valid and
    try again.
```

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets app-debug-androidTest-unaligned.apk.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip app-debug-androidTest-unaligned.apk
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Bei einem gültigen Instrumentierungstestpaket sieht die Ausgabe wie folgt aus:

```
.
|-- AndroidManifest.xml
|--  classes.dex
|-- resources.arsc
|-- LICENSE-junit.txt
|-- junit (directory)
`-- META-INF (directory)
```

Weitere Informationen finden Sie unter Instrumentierung für Android und AWS Device Farm.

INSTRUMENTATION_TEST_PACKAGE_AAPT_DEBUG_BADGING_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

```
We could not extract information about your test package. Please verify that the
      test package is valid by running the command "aapt debug badging <path to your
 test
      package>", and try again after the command does not print any error.
```

Während des Upload-Validierungsprozesses analysiert Device Farm Informationen aus der Ausgabe des `aapt debug badging <path to your package>` Befehls.

Stellen Sie sicher, dass Sie diesen Befehl erfolgreich für Ihr Instrumentierungstestpaket ausführen können.

Im folgenden Beispiel lautet der Name des Pakets app-debug-androidTest-unaligned.apk.

- Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ aapt debug badging app-debug-androidTest-unaligned.apk
```

Bei einem gültigen Instrumentierungstestpaket sieht die Ausgabe wie folgt aus:

```
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'
```

Weitere Informationen finden Sie unter Instrumentierung für Android und AWS Device Farm.

INSTRUMENTATION_TEST_PACKAGE_INSTRUMENTATION_RUNNER_VALU

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

```
We could not find the instrumentation runner value in the AndroidManifest.xml.
      Please verify the test package is valid by running the command "aapt dump xmltree
 <path to
      your test package> AndroidManifest.xml", and try again after finding the
 instrumentation
      runner value behind the keyword "instrumentation."
```

Während des Upload-Validierungsprozesses analysiert Device Farm den Instrumentierungs-Runner-Wert aus dem XML-Analysebaum für eine XML-Datei, die im Paket enthalten ist. Sie können folgenden Befehl verwenden: `aapt dump xmltree <path to your package> AndroidManifest.xml`.

Stellen Sie sicher, dass Sie diesen Befehl für Ihr Instrumentierungstestpaket ausführen und den Instrumentierungswert erfolgreich finden können.

Im folgenden Beispiel lautet der Name des Pakets Test-unaligned.apkapp-debug-android.

- Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ aapt dump xmltree app-debug-androidTest-unaligned.apk AndroidManifest.xml | grep
 -A5 "instrumentation"
```

Bei einem gültigen Instrumentierungstestpaket sieht die Ausgabe wie folgt aus:

```
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
```

Weitere Informationen finden Sie unter Instrumentierung für Android und AWS Device Farm.

INSTRUMENTATION_TEST_PACKAGE_AAPT_DUMP_XMLTREE_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

```
We could not find the valid AndroidManifest.xml in your test package. Please
      verify that the test package is valid by running the command "aapt dump xmltree
 <path to
      your test package> AndroidManifest.xml", and try again after the command does not
 print any
      error.
```

Während der Upload-Validierung analysiert Device Farm mithilfe des folgenden Befehls Informationen aus dem XML-Analysebaum für eine im Paket enthaltene XML-Datei:. `aapt dump xmltree <path to your package> AndroidManifest.xml`

Stellen Sie sicher, dass Sie diesen Befehl erfolgreich für Ihr Instrumentierungstestpaket ausführen können.

Im folgenden Beispiel lautet der Name des Pakets Test-unaligned.apkapp-debug-android.

- Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ aapt dump xmltree app-debug-androidTest-unaligned.apk AndroidManifest.xml
```

Bei einem gültigen Instrumentierungstestpaket sieht die Ausgabe wie folgt aus:

```
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")
```

Weitere Informationen finden Sie unter Instrumentierung für Android und AWS Device Farm.

INSTRUMENTATION_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISS

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

```
We could not find the package name in your test package. Please verify that the
      test package is valid by running the command "aapt debug badging <path to your
 test
      package>", and try again after finding the package name value behind the keyword
 "package:
      name."
```

Während der Upload-Validierung analysiert Device Farm den Wert des Paketnamens aus der Ausgabe des folgenden Befehls:`aapt debug badging <path to your package>`.

Stellen Sie sicher, dass Sie diesen Befehl für Ihr Instrumentierungstestpaket ausführen und den Wert für den Paketnamen erfolgreich finden können.

Im folgenden Beispiel lautet der Name des Pakets app-debug-androidTest-unaligned.apk.

- Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

  ```
  $ aapt debug badging app-debug-androidTest-unaligned.apk | grep "package: name="
  ```

  Bei einem gültigen Instrumentierungstestpaket sieht die Ausgabe wie folgt aus:

  ```
  package: name='com.amazon.aws.adf.android.referenceapp.test' versionCode=''
   versionName='' platformBuildVersionName='5.1.1-1819727'
  ```

  Weitere Informationen finden Sie unter Instrumentierung für Android und AWS Device Farm.

Fehlerbehebung bei iOS-Anwendungstests in AWS Device Farm

Im folgenden Thema werden Fehlermeldungen aufgeführt, die beim Hochladen von iOS-Anwendungstests auftreten können, und Umgehungen für die einzelnen Fehler empfohlen.

> **Note**
>
> Die folgenden Anweisungen gelten für Linux x86_64 and Mac.

IOS_APP_UNZIP_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Ihre Anwendung konnte nicht geöffnet werden. Prüfen Sie, ob die Datei gültig ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Anwendungspaket fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets AWSDeviceFarmi OSReference App.ipa.

1. Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Bei einem gültigen iOS-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

   ```
   .
   `-- Payload (directory)
           `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                       |-- Info.plist
                       `-- (any other files)
   ```

   Weitere Informationen finden Sie unter iOS-Tests in der AWS Device Farm.

IOS_APP_PAYLOAD_DIR_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir konnten das Nutzlastverzeichnis nicht in Ihrer Anwendung finden. Extrahieren Sie Ihre Anwendung, überprüfen Sie, ob das Nutzlastverzeichnis im Paket enthalten ist, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets Farmi App.IPA. AWSDevice OSReference

1. Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Wenn das iOS-Anwendungspaket gültig ist, finden Sie das *Payload* Verzeichnis im Arbeitsverzeichnis.

   ```
   .
   `-- Payload (directory)
           `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                       |-- Info.plist
                       `-- (any other files)
   ```

   Weitere Informationen finden Sie unter iOS-Tests in der AWS Device Farm.

IOS_APP_APP_DIR_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Das .app-Verzeichnis konnte im Nutzlastverzeichnis nicht gefunden werden. Extrahieren Sie Ihre Anwendung und öffnen Sie dann das Nutzlastverzeichnis. Überprüfen Sie, ob sich das .app-Verzeichnis im Verzeichnis befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets AWSDeviceFarmi OSReference App.ipa.

1. Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Wenn das iOS-Anwendungspaket gültig ist, finden Sie innerhalb des *.app Payload* Verzeichnisses ein Verzeichnis wie *AWSDeviceFarmiOSReferenceApp.app* in unserem Beispiel.

   ```
   .
   `-- Payload (directory)
           `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                       |-- Info.plist
                       `-- (any other files)
   ```

   Weitere Informationen finden Sie unter iOS-Tests in der AWS Device Farm.

IOS_APP_PLIST_FILE_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Die Datei Info.plist konnte im .app-Verzeichnis nicht gefunden werden. Extrahieren Sie Ihre Anwendung und öffnen Sie dann das .app-Verzeichnis. Überprüfen Sie, ob sich die Datei Info.plist im Verzeichnis befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets AWSDeviceFarmi OSReference App.ipa.

1. Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Wenn das iOS-Anwendungspaket gültig ist, finden Sie die *Info.plist* Datei wie *AWSDeviceFarmiOSReferenceApp.app* in unserem Beispiel im *.app* Verzeichnis.

   ```
   .
   `-- Payload (directory)
           `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                       |-- Info.plist
                       `-- (any other files)
   ```

   Weitere Informationen finden Sie unter iOS-Tests in der AWS Device Farm.

IOS_APP_CPU_ARCHITECTURE_VALUE_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Der Wert für die CPU-Architektur konnte in der Datei Info.plist nicht gefunden werden. Bitte entpacken Sie Ihre Anwendung und öffnen Sie dann die Datei Info.plist im Verzeichnis .app. Vergewissern Sie sich, dass der Schlüssel "UIRequiredDeviceCapabilities" angegeben ist, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets Farmi App.ipa. AWSDevice OSReference

1. Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Sie sollten die *Info.plist* Datei in einem *.app* Verzeichnis wie *AWSDeviceFarmiOSReferenceApp.app* in unserem Beispiel finden:

   ```
   .
   `-- Payload (directory)
           `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                       |-- Info.plist
                       `-- (any other files)
   ```

3. Um den Wert für die CPU-Architektur zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

   Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

   ```
   $ pip install biplist
   ```

4. Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

   ```
   import biplist
   ```

```
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['UIRequiredDeviceCapabilities']
```

Bei einem gültigen iOS-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

```
['armv7']
```

Weitere Informationen finden Sie unter iOS-Tests in der AWS Device Farm.

IOS_APP_PLATFORM_VALUE_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Der Wert für die Plattform konnte in der Datei Info.plist nicht gefunden werden. Bitte entpacken Sie Ihre Anwendung und öffnen Sie dann die Datei Info.plist im Verzeichnis .app, überprüfen Sie, ob der Schlüssel "CFBundleSupportedPlatforms" angegeben ist, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets Farmi App.ipa. AWSDevice OSReference

1. Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Sie sollten die *Info.plist* Datei in einem *.app* Verzeichnis wie *AWSDeviceFarmiOSReferenceApp.app* in unserem Beispiel finden:

```
.
`-- Payload (directory)
        `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                        |-- Info.plist
                        `-- (any other files)
```

3. Um den Wert für die Plattform zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

   Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

   ```
   $ pip install biplist
   ```

4. Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
   Info.plist')
   print info_plist['CFBundleSupportedPlatforms']
   ```

   Bei einem gültigen iOS-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

   ```
   ['iPhoneOS']
   ```

   Weitere Informationen finden Sie unter iOS-Tests in der AWS Device Farm.

IOS_APP_WRONG_PLATFORM_DEVICE_VALUE

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir haben festgestellt, dass der Wert für das Plattformgerät in der Datei Info.plist falsch war. Bitte entpacken Sie Ihre Anwendung und öffnen Sie dann die Datei Info.plist im Verzeichnis .app. Stellen Sie sicher, dass der Wert des Schlüssels "CFBundleSupportedPlatforms" nicht das Schlüsselwort „simulator“ enthält, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets Farmi App.ipa. AWSDevice OSReference

1. Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Sie sollten die *Info.plist* Datei in einem *.app* Verzeichnis wie *AWSDeviceFarmiOSReferenceApp.app* in unserem Beispiel finden:

   ```
   .
   `-- Payload (directory)
           `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                       |-- Info.plist
                       `-- (any other files)
   ```

3. Um den Wert für die Plattform zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

   Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

   ```
   $ pip install biplist
   ```

4. Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
   Info.plist')
   print info_plist['CFBundleSupportedPlatforms']
   ```

   Bei einem gültigen iOS-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

   ```
   ['iPhoneOS']
   ```

Damit die iOS-Anwendung gültig ist, darf der Wert nicht das Schlüsselwort `simulator` enthalten.

Weitere Informationen finden Sie unter iOS-Tests in der AWS Device Farm.

IOS_APP_FORM_FACTOR_VALUE_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Der Wert für den Formfaktor konnte in der Datei Info.plist nicht gefunden werden. Bitte entpacken Sie Ihre Anwendung und öffnen Sie dann die Datei Info.plist im Verzeichnis .app, überprüfen Sie, ob der Schlüssel "UIDeviceFamily“ angegeben ist, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets Farmi App.ipa. AWSDevice OSReference

1. Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Sie sollten die *Info.plist* Datei in einem *.app* Verzeichnis wie *AWSDeviceFarmiOSReferenceApp.app* in unserem Beispiel finden:

   ```
   .
   `-- Payload (directory)
           `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                       |-- Info.plist
                       `-- (any other files)
   ```

3. Um den Wert für den Formfaktor zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

   Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

   ```
   $ pip install biplist
   ```

4. Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
   Info.plist')
   print info_plist['UIDeviceFamily']
   ```

   Bei einem gültigen iOS-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

   ```
   [1, 2]
   ```

   Weitere Informationen finden Sie unter iOS-Tests in der AWS Device Farm.

IOS_APP_PACKAGE_NAME_VALUE_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Der Wert für den Paketnamen konnte in der Datei Info.plist nicht gefunden werden. Bitte entpacken Sie Ihre Anwendung und öffnen Sie dann die Datei Info.plist im Verzeichnis .app, überprüfen Sie, ob der Schlüssel "CFBundleIdentifier“ angegeben ist, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets Farmi App.ipa. AWSDevice OSReference

1. Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Sie sollten die *Info.plist* Datei in einem *.app* Verzeichnis wie *AWSDeviceFarmiOSReferenceApp.app* in unserem Beispiel finden:

   ```
   .
   `-- Payload (directory)
           `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                       |-- Info.plist
                       `-- (any other files)
   ```

3. Um den Wert für den Paketnamen zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

   Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

   ```
   $ pip install biplist
   ```

4. Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
   Info.plist')
   print info_plist['CFBundleIdentifier']
   ```

   Bei einem gültigen iOS-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

   ```
   Amazon.AWSDeviceFarmiOSReferenceApp
   ```

   Weitere Informationen finden Sie unter iOS-Tests in der AWS Device Farm.

IOS_APP_EXECUTABLE_VALUE_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Der Wert für die ausführbare Datei konnte in der Datei Info.plist nicht gefunden werden. Bitte entpacken Sie Ihre Anwendung und öffnen Sie dann die Datei Info.plist im Verzeichnis .app, überprüfen Sie, ob der Schlüssel "CFBundleExecutable“ angegeben ist, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets Farmi App.ipa. AWSDevice OSReference

1. Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Sie sollten die *Info.plist* Datei in einem *.app* Verzeichnis wie *AWSDeviceFarmiOSReferenceApp.app* in unserem Beispiel finden:

   ```
   .
   `-- Payload (directory)
           `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                       |-- Info.plist
                       `-- (any other files)
   ```

3. Um den Wert für die ausführbare Datei zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

   Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

```
$ pip install biplist
```

4. Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['CFBundleExecutable']
```

Bei einem gültigen iOS-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

```
AWSDeviceFarmiOSReferenceApp
```

Weitere Informationen finden Sie unter iOS-Tests in der AWS Device Farm.

XCTest Tests zur Fehlerbehebung in AWS Device Farm

Im folgenden Thema sind Fehlermeldungen aufgeführt, die beim Hochladen von XCTest Tests auftreten, und es werden Lösungsansätze zur Behebung der einzelnen Fehler empfohlen.

> **Note**
>
> Bei den folgenden Anweisungen wird davon ausgegangen, dass Sie MacOS verwenden.

XCTEST_TEST_PACKAGE_UNZIP_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Ihre ZIP-Datei für den Test konnte nicht geöffnet werden. Prüfen Sie, ob die Datei gültig ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Anwendungspaket fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets .xctest-1.zip. swiftExampleTests

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip swiftExampleTests.xctest-1.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Ein gültiges XCTest Paket sollte eine Ausgabe wie die folgende erzeugen:

   ```
   .
   `-- swiftExampleTests.xctest (directory)
               |-- Info.plist
               `-- (any other files)
   ```

   Weitere Informationen finden Sie unter Integrieren von Device Farm mit XCTest für iOS.

XCTEST_TEST_PACKAGE_XCTEST_DIR_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Wir konnten das .xctest-Verzeichnis nicht in Ihrem Testpaket finden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob das .xctest-Verzeichnis in dem Paket enthalten ist, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets swiftExampleTests.xctest-1.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Wenn das XCTest Paket gültig ist, finden Sie ein Verzeichnis mit einem ähnlichen Namen wie im Arbeitsverzeichnis. *swiftExampleTests.xctest* Der Name sollte mit enden*.xctest*.

   ```
   .
   `-- swiftExampleTests.xctest (directory)
                |-- Info.plist
                `-- (any other files)
   ```

   Weitere Informationen finden Sie unter Integrieren von Device Farm mit XCTest für iOS.

XCTEST_TEST_PACKAGE_PLIST_FILE_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Die Datei Info.plist konnte im .xctest-Verzeichnis nicht gefunden werden. Extrahieren Sie Ihr Testpaket und öffnen Sie dann das .xctest-Verzeichnis. Überprüfen Sie, ob sich die Datei Info.plist im Verzeichnis befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets swiftExampleTests.xctest-1.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip swiftExampleTests.xctest-1.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das XCTest Paket gültig ist, finden Sie die Datei im *Info.plist* Verzeichnis. *.xctest* In unserem Beispiel unten heißt das Verzeichnis*swiftExampleTests.xctest*.

```
.
`-- swiftExampleTests.xctest (directory)
              |-- Info.plist
              `-- (any other files)
```

Weitere Informationen finden Sie unter Integrieren von Device Farm mit XCTest für iOS.

XCTEST_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Der Wert für den Paketnamen konnte in der Datei Info.plist nicht gefunden werden. Bitte entpacken Sie Ihr Testpaket und öffnen Sie dann die Datei Info.plist, überprüfen Sie, ob der Schlüssel "CFBundleIdentifier“ angegeben ist, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets .xctest-1.zip. swiftExampleTests

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip swiftExampleTests.xctest-1.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

Sie sollten die *Info.plist* Datei in einem *.xctest* Verzeichnis wie in unserem Beispiel finden: *swiftExampleTests.xctest*

```
.
`-- swiftExampleTests.xctest (directory)
              |-- Info.plist
              `-- (any other files)
```

3. Um den Wert für den Paketnamen zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

   Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

   ```
   $ pip install biplist
   ```

4. Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

   ```
   import biplist
   info_plist = biplist.readPlist('swiftExampleTests.xctest/Info.plist')
   print info_plist['CFBundleIdentifier']
   ```

   Ein gültiges XCtest Anwendungspaket sollte eine Ausgabe wie die folgende erzeugen:

   ```
   com.amazon.kanapka.swiftExampleTests
   ```

   Weitere Informationen finden Sie unter Integrieren von Device Farm mit XCTest für iOS.

XCTEST_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

> **Warning**
>
> Der Wert für die ausführbare Datei konnte in der Datei Info.plist nicht gefunden werden. Bitte entpacken Sie Ihr Testpaket und öffnen Sie dann die Datei Info.plist, überprüfen Sie, ob der Schlüssel "CFBundleExecutable“ angegeben ist, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets .xctest-1.zip. swiftExampleTests

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

   Sie sollten die *Info.plist* Datei in einem *.xctest* Verzeichnis wie in unserem Beispiel finden: *swiftExampleTests.xctest*

```
.
`-- swiftExampleTests.xctest (directory)
              |-- Info.plist
              `-- (any other files)
```

3. Um den Wert für den Paketnamen zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

   Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

```
$ pip install biplist
```

4. Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

```
import biplist
info_plist = biplist.readPlist('swiftExampleTests.xctest/Info.plist')
print info_plist['CFBundleExecutable']
```

   Ein gültiges XCtest Anwendungspaket sollte eine Ausgabe wie die folgende erzeugen:

```
swiftExampleTests
```

   Weitere Informationen finden Sie unter Integrieren von Device Farm mit XCTest für iOS.

Fehlerbehebung bei XCTest UI-Tests in AWS Device Farm

Im folgenden Thema werden Fehlermeldungen aufgeführt, die beim Hochladen von XCTest UI-Tests auftreten, und es werden Lösungsansätze zur Behebung der einzelnen Fehler empfohlen.

> **Note**
>
> Die folgenden Anweisungen gelten für Linux x86_64 and Mac.

XCTEST_UI_TEST_PACKAGE_UNZIP_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

```
We could not open your test IPA file. Please verify that the file is valid and try again.
```

Stellen Sie sicher, dass Sie das Anwendungspaket fehlerfrei dekomprimieren können. Im folgenden Beispiel ist der Name des Pakets swift-sample-UI.ipa.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Bei einem gültigen iOS-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                       |-- Info.plist
   ```

```
                    |                       `-- (any other files)
                    `-- (any other files)
```

Weitere Informationen finden Sie unter Integrieren der XCTest Benutzeroberfläche für iOS mit Device Farm.

XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

`We could not find the Payload directory inside your test package. Please unzip your test package, verify that the Payload directory is inside the package, and try again.`

Im folgenden Beispiel ist der Name des Pakets swift-sample-UI.ipa.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Wenn das XCTest UI-Paket gültig ist, finden Sie das *Payload* Verzeichnis im Arbeitsverzeichnis.

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                       |-- Info.plist
                       |                       `-- (any other files)
                       `-- (any other files)
   ```

Weitere Informationen finden Sie unter Integrieren der XCTest Benutzeroberfläche für iOS mit Device Farm.

XCTEST_UI_TEST_PACKAGE_APP_DIR_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

```
We could not find the .app directory inside the Payload directory. Please unzip your test package and then open the Payload directory, verify that the .app directory is inside the directory, and try again.
```

Im folgenden Beispiel ist der Name des Pakets swift-sample-UI.ipa.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Wenn das XCTest UI-Paket gültig ist, finden Sie innerhalb des *.app* *Payload* Verzeichnisses ein Verzeichnis wie *swift-sampleUITests-Runner.app* in unserem Beispiel.

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                       |-- Info.plist
                       |                       `-- (any other files)
                       `-- (any other files)
   ```

   Weitere Informationen finden Sie unter Integrieren der XCTest Benutzeroberfläche für iOS mit Device Farm.

XCTEST_UI_TEST_PACKAGE_PLUGINS_DIR_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

```
We could not find the Plugins directory inside the .app directory. Please unzip your test package and then open the .app directory, verify that the Plugins directory is inside the directory, and try again.
```

Im folgenden Beispiel ist der Name des Pakets swift-sample-UI.ipa.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Wenn das XCTest UI-Paket gültig ist, finden Sie das *Plugins* Verzeichnis in einem *.app* Verzeichnis. In unserem Beispiel heißt das Verzeichnis*swift-sampleUITests-Runner.app*.

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                       |-- Info.plist
                       |                       `-- (any other files)
                       `-- (any other files)
   ```

   Weitere Informationen finden Sie unter Integrieren der XCTest Benutzeroberfläche für iOS mit Device Farm.

XCTEST_UI_TEST_PACKAGE_XCTEST_DIR_MISSING_IN_PLUGINS_DIR

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

```
We could not find the .xctest directory inside the plugins directory.
Please unzip your test package and then open the plugins directory, verify
that the .xctest directory is inside the directory, and try again.
```

Im folgenden Beispiel ist der Name des Pakets swift-sample-UI.ipa.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Wenn das XCTest UI-Paket gültig ist, finden Sie ein *.xctest* Verzeichnis innerhalb des *Plugins* Verzeichnisses. In unserem Beispiel heißt das Verzeichnis*swift-sampleUITests.xctest*.

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                         |-- Info.plist
                       |                         `-- (any other files)
                       `-- (any other files)
   ```

   Weitere Informationen finden Sie unter Integrieren der XCTest Benutzeroberfläche für iOS mit Device Farm.

XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

```
We could not find the Info.plist file inside the .app directory. Please
unzip your test package and then open the .app directory, verify that the
Info.plist file is inside the directory, and try again.
```

Im folgenden Beispiel ist der Name des Pakets swift-sample-UI.ipa.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Wenn das XCTest UI-Paket gültig ist, finden Sie die *`Info.plist`* Datei im *`.app`* Verzeichnis. In unserem Beispiel unten heißt das Verzeichnis*`swift-sampleUITests-Runner.app`*.

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                       |-- Info.plist
                       |                       `-- (any other files)
                       `-- (any other files)
   ```

   Weitere Informationen finden Sie unter Integrieren der XCTest Benutzeroberfläche für iOS mit Device Farm.

XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING_IN_XCTEST_DIR

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

```
We could not find the Info.plist file inside the .xctest directory. Please unzip your test package and then open the .xctest directory, verify that the Info.plist file is inside the directory, and try again.
```

Im folgenden Beispiel ist der Name des Pakets swift-sample-UI.ipa.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Wenn das XCTest UI-Paket gültig ist, finden Sie die *Info.plist* Datei im *.xctest* Verzeichnis. In unserem Beispiel unten heißt das Verzeichnis*swift-sampleUITests.xctest*.

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                         |-- Info.plist
                       |                         `-- (any other files)
                       `-- (any other files)
   ```

   Weitere Informationen finden Sie unter Integrieren der XCTest Benutzeroberfläche für iOS mit Device Farm.

XCTEST_UI_TEST_PACKAGE_CPU_ARCHITECTURE_VALUE_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

```
We could not the CPU architecture value in the Info.plist file. Please
unzip your test package and then open the Info.plist file inside the .app
directory, verify that the key "UIRequiredDeviceCapabilities" is specified,
and try again.
```

Im folgenden Beispiel ist der Name des Pakets swift-sample-UI.ipa.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Sie sollten die *Info.plist* Datei in einem *.app* Verzeichnis wie *swift-sampleUITests-Runner.app* in unserem Beispiel finden:

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                       |-- Info.plist
                       |                       `-- (any other files)
                       `-- (any other files)
   ```

3. Um den Wert für die CPU-Architektur zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

   Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

```
$ pip install biplist
```

4. Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/
Info.plist')
print info_plist['UIRequiredDeviceCapabilities']
```

Ein gültiges XCtest UI-Paket sollte eine Ausgabe wie die folgende erzeugen:

```
['armv7']
```

Weitere Informationen finden Sie unter Integrieren der XCTest Benutzeroberfläche für iOS mit Device Farm.

XCTEST_UI_TEST_PACKAGE_PLATFORM_VALUE_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

```
We could not find the platform value in the Info.plist. Please unzip your
test package and then open the Info.plist file inside the .app directory,
verify that the key "CFBundleSupportedPlatforms" is specified, and try
again.
```

Im folgenden Beispiel ist der Name des Pakets swift-sample-UI.ipa.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip swift-sample-UI.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Sie sollten die *Info.plist* Datei in einem *.app* Verzeichnis wie *swift-sampleUITests-Runner.app* in unserem Beispiel finden:

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                        |-- Info.plist
                        |-- Plugins (directory)
                        |       `swift-sampleUITests.xctest (directory)
                        |                         |-- Info.plist
                        |                         `-- (any other files)
                        `-- (any other files)
```

3. Um den Wert für die Plattform zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

   Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

   ```
   $ pip install biplist
   ```

4. Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
   print info_plist['CFBundleSupportedPlatforms']
   ```

   Ein gültiges XCtest UI-Paket sollte eine Ausgabe wie die folgende erzeugen:

   ```
   ['iPhoneOS']
   ```

   Weitere Informationen finden Sie unter Integrieren der XCTest Benutzeroberfläche für iOS mit Device Farm.

XCTEST_UI_TEST_PACKAGE_WRONG_PLATFORM_DEVICE_VALUE

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

```
We found the platform device value was wrong in the Info.plist file. Please
unzip your test package and then open the Info.plist file inside the .app
directory, verify that the value of the key "CFBundleSupportedPlatforms"
does not contain the keyword "simulator", and try again.
```

Im folgenden Beispiel ist der Name des Pakets swift-sample-UI.ipa.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Sie sollten die *Info.plist* Datei in einem *.app* Verzeichnis wie *swift-sampleUITests-Runner.app* in unserem Beispiel finden:

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                       |-- Info.plist
                       |                       `-- (any other files)
                       `-- (any other files)
   ```

3. Um den Wert für die Plattform zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

   Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

   ```
   $ pip install biplist
   ```

4. Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
   ```

```
print info_plist['CFBundleSupportedPlatforms']
```

Ein gültiges XCtest UI-Paket sollte eine Ausgabe wie die folgende erzeugen:

```
['iPhoneOS']
```

Wenn das XCTest UI-Paket gültig ist, sollte der Wert das Schlüsselwort nicht enthalten`simulator`.

Weitere Informationen finden Sie unter Integrieren der XCTest Benutzeroberfläche für iOS mit Device Farm.

XCTEST_UI_TEST_PACKAGE_FORM_FACTOR_VALUE_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

```
We could not the form factor value in the Info.plist. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the key "UIDeviceFamily" is specified, and try again.
```

Im folgenden Beispiel ist der Name des Pakets swift-sample-UI.ipa.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Sie sollten die *`Info.plist`* Datei in einem *`.app`* Verzeichnis wie *`swift-sampleUITests-Runner.app`* in unserem Beispiel finden:

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
   ```

```
                        |-- Info.plist
                        |-- Plugins (directory)
                        |       `swift-sampleUITests.xctest (directory)
                        |                        |-- Info.plist
                        |                        `-- (any other files)
                        `-- (any other files)
```

3. Um den Wert für den Formfaktor zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

   Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

   ```
   $ pip install biplist
   ```

4. Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
   print info_plist['UIDeviceFamily']
   ```

   Ein gültiges XCtest UI-Paket sollte eine Ausgabe wie die folgende erzeugen:

   ```
   [1, 2]
   ```

   Weitere Informationen finden Sie unter Integrieren der XCTest Benutzeroberfläche für iOS mit Device Farm.

XCTEST_UI_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

```
We could not find the package name value in the Info.plist file. Please
unzip your test package and then open the Info.plist file inside the .app
directory, verify that the key "CFBundleIdentifier" is specified, and try
again.
```

Im folgenden Beispiel ist der Name des Pakets swift-sample-UI.ipa.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip swift-sample-UI.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

   Sie sollten die *Info.plist* Datei in einem *.app* Verzeichnis wie *swift-sampleUITests-Runner.app* in unserem Beispiel finden:

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                        |-- Info.plist
                        |-- Plugins (directory)
                        |       `swift-sampleUITests.xctest (directory)
                        |                         |-- Info.plist
                        |                         `-- (any other files)
                        `-- (any other files)
```

3. Um den Wert für den Paketnamen zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

   Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

```
$ pip install biplist
```

4. Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['CFBundleIdentifier']
```

   Ein gültiges XCtest UI-Paket sollte eine Ausgabe wie die folgende erzeugen:

```
com.apple.test.swift-sampleUITests-Runner
```

Weitere Informationen finden Sie unter Integrieren der XCTest Benutzeroberfläche für iOS mit Device Farm.

XCTEST_UI_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

```
We could not find the executable value in the Info.plist file. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the key "CFBundleExecutable" is specified, and try again.
```

Im folgenden Beispiel ist der Name des Pakets swift-sample-UI.ipa.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Sie sollten die *Info.plist* Datei in einem *.app* Verzeichnis wie *swift-sampleUITests-Runner.app* in unserem Beispiel finden:

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                       |-- Info.plist
                       |                       `-- (any other files)
                       `-- (any other files)
   ```

3. Um den Wert für die ausführbare Datei zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

   Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

   ```
   $ pip install biplist
   ```

4. Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
   print info_plist['CFBundleExecutable']
   ```

   Ein gültiges XCtest UI-Paket sollte eine Ausgabe wie die folgende erzeugen:

   ```
   XCTRunner
   ```

   Weitere Informationen finden Sie unter Integrieren der XCTest Benutzeroberfläche für iOS mit Device Farm.

XCTEST_UI_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

```
We could not find the package name value in the Info.plist file inside
the .xctest directory. Please unzip your test package and then open
the Info.plist file inside the .xctest directory, verify that the key
"CFBundleIdentifier" is specified, and try again.
```

Im folgenden Beispiel ist der Name des Pakets swift-sample-UI.ipa.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Sie sollten die *Info.plist* Datei in einem *.app* Verzeichnis wie *swift-sampleUITests-Runner.app* in unserem Beispiel finden:

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                        |-- Info.plist
                        |-- Plugins (directory)
                        |       `swift-sampleUITests.xctest (directory)
                        |                         |-- Info.plist
                        |                         `-- (any other files)
                        `-- (any other files)
```

3. Um den Wert für den Paketnamen zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

   Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

   ```
   $ pip install biplist
   ```

4. Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Plugins/
   swift-sampleUITests.xctest/Info.plist')
   print info_plist['CFBundleIdentifier']
   ```

   Ein gültiges XCtest UI-Paket sollte eine Ausgabe wie die folgende erzeugen:

   ```
   com.amazon.swift-sampleUITests
   ```

   Weitere Informationen finden Sie unter Integrieren der XCTest Benutzeroberfläche für iOS mit Device Farm.

XCTEST_UI_TEST_PACKAGE_TEST_EXECUTABLE_VALUE_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

```
We could not find the executable value in the Info.plist file inside
the .xctest directory. Please unzip your test package and then open
the Info.plist file inside the .xctest directory, verify that the key
"CFBundleExecutable" is specified, and try again.
```

Im folgenden Beispiel ist der Name des Pakets swift-sample-UI.ipa.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Sie sollten die *Info.plist* Datei in einem *.app* Verzeichnis wie *swift-sampleUITests-Runner.app* in unserem Beispiel finden:

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                       |-- Info.plist
                       |                       `-- (any other files)
                       `-- (any other files)
   ```

3. Um den Wert für die ausführbare Datei zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

   Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

```
$ pip install biplist
```

4. Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Plugins/
swift-sampleUITests.xctest/Info.plist')
print info_plist['CFBundleExecutable']
```

Ein gültiges XCtest UI-Paket sollte eine Ausgabe wie die folgende erzeugen:

```
swift-sampleUITests
```

Weitere Informationen finden Sie unter Integrieren der XCTest Benutzeroberfläche für iOS mit Device Farm.

XCTEST_UI_TEST_PACKAGE_MULTIPLE_APP_DIRS

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

`We found multiple .app directories inside your test package. Please unzip your test package, verify that only a single .app directory is present inside the package, then try again.`

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip swift-sample-UI.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das XCTest UI-Paket gültig ist, sollten Sie nur ein einziges Verzeichnis wie in unserem Beispiel im .zip-Testpaket finden. `.app` `swift-sampleUITests-Runner.app`

```
.
`--swift-sample-UI.zip--(directory)
    `-- swift-sampleUITests-Runner.app (directory)
            |-- Info.plist
            |-- Plugins (directory)
            |       `swift-sampleUITests.xctest (directory)
           |            |-- Info.plist
           |            `-- (any other files)
            `-- (any other files)
    `-- (any other files)
```

Weitere Informationen finden Sie unter Integrieren der XCTest Benutzeroberfläche für iOS mit Device Farm.

XCTEST_UI_TEST_PACKAGE_MULTIPLE_IPA_DIRS

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

`We found multiple .ipa directories inside your test package. Please unzip your test package, verify that only a single .ipa directory is present inside the package, then try again.`

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip swift-sample-UI.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Wenn das XCTest UI-Paket gültig ist, sollten Sie nur ein einziges Verzeichnis wie in unserem Beispiel innerhalb des .zip-Testpakets finden. `.ipa sampleUITests.ipa`

   ```
   .
   `--swift-sample-UI.zip--(directory)
       `-- sampleUITests.ipa (directory)
   ```

```
            `-- Payload (directory)
                `-- swift-sampleUITests-Runner.app (directory)
    `-- (any other files)
```

Weitere Informationen finden Sie unter Integrieren der XCTest Benutzeroberfläche für iOS mit Device Farm.

XCTEST_UI_TEST_PACKAGE_BOTH_APP_AND_IPA_DIR_PRESENT

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

`We found both .app and .ipa files inside your test package. Please unzip your test package, verify that only a single .app or .ipa file is present inside the package, then try again.`

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip swift-sample-UI.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Wenn das XCTest UI-Paket gültig ist, sollten Sie entweder das Verzeichnis like oder das Verzeichnis wie in unserem Beispiel innerhalb des .zip-Testpakets finden. `.ipa` `sampleUITests.ipa` `.app` `swift-sampleUITests-Runner.app` Ein Beispiel für ein gültiges XCTEST_UI-Testpaket finden Sie in unserer Dokumentation unter. Integrieren der XCTest Benutzeroberfläche für iOS mit Device Farm

   ```
   .
   `--swift-sample-UI.zip--(directory)
       `-- sampleUITests.ipa (directory)
               `-- Payload (directory)
                   `-- swift-sampleUITests-Runner.app (directory)
      `-- (any other files)
   ```

or

```
.
`--swift-sample-UI.zip--(directory)
    `-- swift-sampleUITests-Runner.app (directory)
            |-- Info.plist
            |-- Plugins (directory)
            `-- (any other files)
   `-- (any other files)
```

Weitere Informationen finden Sie unter Integrieren der XCTest Benutzeroberfläche für iOS mit Device Farm.

XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_PRESENT_IN_ZIP

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

`We found a Payload directory inside your .zip test package. Please unzip your test package, ensure that a Payload directory is not present in the package, then try again.`

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

   ```
   $ unzip swift-sample-UI.zip
   ```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

   ```
   $ tree .
   ```

   Wenn das UI-Paket gültig ist, sollten Sie in Ihrem Testpaket kein Payload-Verzeichnis finden. XCTest

   ```
   .
   `--swift-sample-UI.zip--(directory)
       `-- swift-sampleUITests-Runner.app (directory)
               |-- Info.plist
   ```

```
            |-- Plugins (directory)
            `-- (any other files)
    `-- Payload (directory) [This directory should not be present]
            |-- (any other files)
    `-- (any other files)
```

Weitere Informationen finden Sie unter Integrieren der XCTest Benutzeroberfläche für iOS mit Device Farm.

Sicherheit in AWS Device Farm

Cloud-Sicherheit AWS hat höchste Priorität. Als AWS Kunde profitieren Sie von einer Rechenzentrums- und Netzwerkarchitektur, die darauf ausgelegt sind, die Anforderungen der sicherheitssensibelsten Unternehmen zu erfüllen.

Sicherheit ist eine gemeinsame Verantwortung von Ihnen AWS und Ihnen. Das Modell der geteilten Verantwortung beschreibt dies als Sicherheit der Cloud und Sicherheit in der Cloud:

- Sicherheit der Cloud — AWS ist verantwortlich für den Schutz der Infrastruktur, die AWS Dienste in der AWS Cloud ausführt. AWS bietet Ihnen auch Dienste, die Sie sicher nutzen können. Externe Prüfer testen und verifizieren regelmäßig die Wirksamkeit unserer Sicherheitsmaßnahmen im Rahmen der AWS . Weitere Informationen zu den geltenden Compliance-Programmen finden Sie unter AWS-Services in Umfang nach Compliance-Programm . AWS Device Farm
- Sicherheit in der Cloud – Ihr Verantwortungsumfang wird durch den AWS -Dienst bestimmt, den Sie verwenden. Sie sind auch für andere Faktoren verantwortlich, etwa für die Vertraulichkeit Ihrer Daten, für die Anforderungen Ihres Unternehmens und für die geltenden Gesetze und Vorschriften.

Diese Dokumentation hilft Ihnen zu verstehen, wie Sie das Modell der gemeinsamen Verantwortung bei der Verwendung von Device Farm anwenden. In den folgenden Themen erfahren Sie, wie Sie Device Farm konfigurieren, um Ihre Sicherheits- und Compliance-Ziele zu erreichen. Sie erfahren auch, wie Sie andere AWS-Services nutzen können, mit denen Sie Ihre Device Farm Farm-Ressourcen überwachen und sichern können.

Topics

Identitäts- und Zugriffsmanagement in AWS Device Farm

Zielgruppe

Wie Sie AWS Identity and Access Management (IAM) verwenden, hängt von Ihrer Rolle ab:

- Servicebenutzer – Fordern Sie von Ihrem Administrator Berechtigungen an, wenn Sie nicht auf Features zugreifen können (siehe Fehlerbehebung bei Identität und Zugriff auf AWS Device Farm).
- Serviceadministrator – Bestimmen Sie den Benutzerzugriff und stellen Sie Berechtigungsanfragen (siehe So funktioniert AWS Device Farm mit IAM).
- IAM-Administrator – Schreiben Sie Richtlinien zur Zugriffsverwaltung (siehe Beispiele für identitätsbasierte Richtlinien von AWS Device Farm).

Authentifizierung mit Identitäten

Authentifizierung ist die Art und Weise, wie Sie sich AWS mit Ihren Identitätsdaten anmelden. Sie müssen sich als IAM-Benutzer authentifizieren oder eine IAM-Rolle annehmen. Root-Benutzer des AWS-Kontos

Sie können sich als föderierte Identität anmelden, indem Sie Anmeldeinformationen aus einer Identitätsquelle wie AWS IAM Identity Center (IAM Identity Center), Single Sign-On-Authentifizierung oder Anmeldeinformationen verwenden. Google/Facebook Weitere Informationen zum Anmelden finden Sie unter So melden Sie sich bei Ihrem AWS-Konto an im Benutzerhandbuch für AWS-Anmeldung .

AWS Bietet für den programmatischen Zugriff ein SDK und eine CLI zum kryptografischen Signieren von Anfragen. Weitere Informationen finden Sie unter AWS Signature Version 4 for API requests im IAM-Benutzerhandbuch.

AWS-Konto Root-Benutzer

Wenn Sie einen erstellen AWS-Konto, beginnen Sie mit einer Anmeldeidentität, dem sogenannten AWS-Konto Root-Benutzer, der vollständigen Zugriff auf alle AWS-Services Ressourcen hat. Wir raten ausdrücklich davon ab, den Root-Benutzer für Alltagsaufgaben zu verwenden. Eine Liste der Aufgaben, für die Sie sich als Root-Benutzer anmelden müssen, finden Sie unter Tasks that require root user credentials im IAM-Benutzerhandbuch.

IAM-Benutzer und -Gruppen

Ein IAM-Benutzer ist eine Identität mit bestimmten Berechtigungen für eine einzelne Person oder Anwendung. Wir empfehlen die Verwendung temporärer Anmeldeinformationen anstelle von IAM-Benutzern mit langfristigen Anmeldeinformationen. Weitere Informationen finden Sie im IAM-Benutzerhandbuch unter Erfordern, dass menschliche Benutzer für den Zugriff AWS mithilfe temporärer Anmeldeinformationen einen Verbund mit einem Identitätsanbieter verwenden müssen.

Eine IAM-Gruppe spezifiziert eine Sammlung von IAM-Benutzern und erleichtert die Verwaltung von Berechtigungen für große Gruppen von Benutzern. Weitere Informationen finden Sie unter Anwendungsfälle für IAM-Benutzer im IAM-Benutzerhandbuch.

IAM-Rollen

Eine IAM-Rolle ist eine Identität mit spezifischen Berechtigungen, die temporäre Anmeldeinformationen bereitstellt. Sie können eine Rolle übernehmen, indem Sie von einer Benutzer- zu einer IAM-Rolle (Konsole) wechseln AWS CLI oder einen AWS API-Vorgang aufrufen. Weitere Informationen finden Sie unter Methoden, um eine Rolle zu übernehmen im IAM-Benutzerhandbuch.

IAM-Rollen sind nützlich für den Verbundbenutzer-Zugriff, temporäre IAM-Benutzerberechtigungen, kontoübergreifenden Zugriff, serviceübergreifenden Zugriff und Anwendungen, die auf Amazon EC2 laufen. Weitere Informationen finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

So funktioniert AWS Device Farm mit IAM

Bevor Sie IAM verwenden, um den Zugriff auf Device Farm zu verwalten, sollten Sie wissen, welche IAM-Funktionen für die Verwendung mit Device Farm verfügbar sind. Einen allgemeinen Überblick darüber, wie Device Farm und andere AWS Dienste mit IAM funktionieren, finden Sie unter AWS Services That Work with IAM im IAM-Benutzerhandbuch.

Themen

- Identitätsbasierte Richtlinien für Device Farm
- Ressourcenbasierte Richtlinien für Device Farm
- Zugriffskontrolllisten
- Autorisierung basierend auf Device Farm Farm-Tags
- IAM-Rollen für Device Farm

Identitätsbasierte Richtlinien für Device Farm

Mit identitätsbasierten IAM-Richtlinien können Sie angeben, welche Aktionen und Ressourcen erteilt oder abgelehnt werden. Darüber hinaus können Sie die Bedingungen festlegen, unter denen Aktionen zugelassen oder abgelehnt werden. Device Farm unterstützt bestimmte Aktionen, Ressourcen und Bedingungsschlüssel. Informationen zu sämtlichen Elementen, die Sie in einer JSON-Richtlinie verwenden, finden Sie in der IAM-Referenz für JSON-Richtlinienelemente im IAM-Benutzerhandbuch.

Aktionen

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer auf was Zugriff hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das Element `Action` einer JSON-Richtlinie beschreibt die Aktionen, mit denen Sie den Zugriff in einer Richtlinie zulassen oder verweigern können. Nehmen Sie Aktionen in eine Richtlinie auf, um Berechtigungen zur Ausführung des zugehörigen Vorgangs zu erteilen.

Richtlinienaktionen in Device Farm verwenden vor der Aktion das folgende Präfix:`devicefarm:`. Um beispielsweise jemandem die Erlaubnis zu erteilen, Selenium-Sitzungen mit dem `CreateTestGridUrl` API-Vorgang zum Testen des Desktop-Browsers Device Farm zu starten, nehmen Sie die `devicefarm:CreateTestGridUrl` Aktion in die Richtlinie auf. Richtlinienanweisungen müssen entweder ein – `Action`oder ein `NotAction`-Element enthalten. Device Farm definiert eigene Aktionen, die Aufgaben beschreiben, die Sie mit diesem Dienst ausführen können.

Um mehrere Aktionen in einer einzigen Anweisung anzugeben, trennen Sie sie wie folgt durch Kommata:

```
"Action": [
      "devicefarm:action1",
      "devicefarm:action2"
```

Sie können auch Platzhalter verwenden, um mehrere Aktionen anzugeben. Beispielsweise können Sie alle Aktionen festlegen, die mit dem Wort `List` beginnen, einschließlich der folgenden Aktion:

```
"Action": "devicefarm:List*"
```

Eine Liste der Gerätefarmaktionen finden Sie unter Aktionen definiert von AWS Device Farm in der IAM Service Authorization Reference.

Ressourcen

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das JSON-Richtlinienelement `Resource` gibt die Objekte an, auf welche die Aktion angewendet wird. Als Best Practice geben Sie eine Ressource mit dem zugehörigen Amazon-Ressourcennamen (ARN) an. Verwenden Sie für Aktionen, die keine Berechtigungen auf Ressourcenebene unterstützen, einen Platzhalter (*), um anzugeben, dass die Anweisung für alle Ressourcen gilt.

```
"Resource": "*"
```

Die Amazon EC2 EC2-Instance-Ressource hat den folgenden ARN:

```
arn:${Partition}:ec2:${Region}:${Account}:instance/${InstanceId}
```

Weitere Informationen zum Format von ARNs finden Sie unter Amazon Resource Names (ARNs) und AWS Service Namespaces.

Wenn Sie beispielsweise die `i-1234567890abcdef0`-Instance in Ihrer Anweisung angeben möchten, verwenden Sie den folgenden ARN:

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:instance/i-1234567890abcdef0"
```

Um alle Instances anzugeben, die zu einem Konto gehören, verwenden Sie den Platzhalter (*):

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:instance/*"
```

Einige Device Farm Farm-Aktionen, z. B. zum Erstellen von Ressourcen, können für eine Ressource nicht ausgeführt werden. In diesen Fällen müssen Sie den Platzhalter (*) verwenden.

```
"Resource": "*"
```

Viele Amazon EC2-API-Aktionen umfassen mehrere Ressourcen. Zum Beispiel wird mit `AttachVolume` einer Instance ein Amazon EBS-Volume angefügt, sodass der IAM-Benutzer über Berechtigungen zur Verwendung des Volumes und der Instance verfügen muss. Um mehrere Ressourcen in einer einzigen Anweisung anzugeben, trennen Sie sie ARNs durch Kommas.

```
"Resource": [
      "resource1",
      "resource2"
```

Eine Liste der Device Farm Farm-Ressourcentypen und ihrer ARNs Eigenschaften finden Sie unter Ressourcentypen definiert von AWS Device Farm in der IAM Service Authorization Reference. Informationen darüber, mit welchen Aktionen Sie den ARN der einzelnen Ressourcen angeben können, finden Sie unter Actions defined by AWS Device Farm in der IAM Service Authorization Reference.

Bedingungsschlüssel

Administratoren können mithilfe von AWS JSON-Richtlinien angeben, wer Zugriff auf was hat. Das heißt, welcher Prinzipal Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen kann.

Das Element `Condition` gibt an, wann Anweisungen auf der Grundlage definierter Kriterien ausgeführt werden. Sie können bedingte Ausdrücke erstellen, die Bedingungsoperatoren verwenden, z. B. ist gleich oder kleiner als, damit die Bedingung in der Richtlinie mit Werten in der Anforderung übereinstimmt. Eine Übersicht aller AWS globalen Bedingungsschlüssel finden Sie unter Kontextschlüssel für AWS globale Bedingungen im IAM-Benutzerhandbuch.

Device Farm definiert seinen eigenen Satz von Bedingungsschlüsseln und unterstützt auch die Verwendung einiger globaler Bedingungsschlüssel. Eine Übersicht aller AWS globalen Bedingungsschlüssel finden Sie unter AWS Globale Bedingungskontextschlüssel im IAM-Benutzerhandbuch.

Eine Liste der Device Farm Farm-Bedingungsschlüssel finden Sie unter Bedingungsschlüssel für AWS Device Farm in der IAM Service Authorization Reference. Informationen zu den Aktionen und Ressourcen, mit denen Sie einen Bedingungsschlüssel verwenden können, finden Sie AWS Device Farm in der IAM Service Authorization Reference unter Actions defined by.

Beispiele

Beispiele für identitätsbasierte Device Farm Farm-Richtlinien finden Sie unter. Beispiele für identitätsbasierte Richtlinien von AWS Device Farm

Ressourcenbasierte Richtlinien für Device Farm

Device Farm unterstützt keine ressourcenbasierten Richtlinien.

Zugriffskontrolllisten

Device Farm unterstützt keine Zugriffskontrolllisten (ACLs).

Autorisierung basierend auf Device Farm Farm-Tags

Sie können Tags an Device Farm-Ressourcen anhängen oder Tags in einer Anfrage an Device Farm übergeben. Um den Zugriff auf der Grundlage von Tags zu steuern, geben Sie im Bedingungselement einer Richtlinie Tag-Informationen an, indem Sie die Schlüssel `aws:ResourceTag/`*`key-name`*, `aws:RequestTag/`*`key-name`*, oder Bedingung `aws:TagKeys` verwenden. Weitere Informationen zum Taggen von Gerätefarm-Ressourcen finden Sie unterTaggen in der Device Farm.

Ein Beispiel für eine identitätsbasierte Richtlinie zur Einschränkung des Zugriffs auf eine Ressource auf der Grundlage der Markierungen dieser Ressource finden Sie unter Anzeigen von Device Farm Farm-Desktop-Browser-Testprojekten auf der Grundlage von Tags.

IAM-Rollen für Device Farm

Eine IAM-Rolle ist eine Entität in Ihrem AWS Konto, die über bestimmte Berechtigungen verfügt.

Temporäre Anmeldeinformationen mit Device Farm verwenden

Device Farm unterstützt die Verwendung temporärer Anmeldeinformationen.

Sie können temporäre Anmeldeinformationen verwenden, um sich bei Federation anzumelden und eine IAM-Rolle oder eine kontoübergreifende Rolle zu übernehmen. Sie erhalten temporäre Sicherheitsanmeldedaten, indem Sie AWS STS API-Operationen wie oder aufrufen. AssumeRoleGetFederationToken

Service-verknüpfte Rollen

Mit dienstbezogenen Rollen können AWS Dienste auf Ressourcen in anderen Diensten zugreifen, um eine Aktion in Ihrem Namen auszuführen. Serviceverknüpfte Rollen werden in Ihrem IAM-

Konto angezeigt und gehören zum Service. Ein IAM-Administrator kann die Berechtigungen für dienstbezogene Rollen anzeigen, aber nicht bearbeiten.

Device Farm verwendet dienstverknüpfte Rollen in der Device Farm Farm-Desktop-Browser-Testfunktion. Informationen zu diesen Rollen finden Sie im Entwicklerhandbuch unter Verwenden von dienstverknüpften Rollen beim Testen von Device Farm Farm-Desktopbrowsern.

Servicerollen

Device Farm unterstützt keine Dienstrollen.

Dieses Feature ermöglicht einem Service das Annehmen einer Servicerolle in Ihrem Namen. Diese Rolle gewährt dem Service Zugriff auf Ressourcen in anderen Diensten, um eine Aktion in Ihrem Namen auszuführen. Servicerollen werden in Ihrem IAM-Konto angezeigt und gehören zum Konto. Dies bedeutet, dass ein IAM-Administrator die Berechtigungen für diese Rolle ändern kann. Dies kann jedoch die Funktionalität des Dienstes beeinträchtigen.

Verwalten des Zugriffs mit Richtlinien

Sie kontrollieren den Zugriff, AWS indem Sie Richtlinien erstellen und diese an AWS Identitäten oder Ressourcen anhängen. Eine Richtlinie definiert Berechtigungen, wenn sie mit einer Identität oder Ressource verknüpft sind. AWS bewertet diese Richtlinien, wenn ein Principal eine Anfrage stellt. Die meisten Richtlinien werden AWS als JSON-Dokumente gespeichert. Weitere Informationen zu JSON-Richtliniendokumenten finden Sie unter Übersicht über JSON-Richtlinien im IAM-Benutzerhandbuch.

Mit Hilfe von Richtlinien legen Administratoren fest, wer Zugriff auf was hat, indem sie definieren, welches Prinzipal welche Aktionen auf welchen Ressourcenund unter welchen Bedingungendurchführen darf.

Standardmäßig haben Benutzer, Gruppen und Rollen keine Berechtigungen. Ein IAM-Administrator erstellt IAM-Richtlinien und fügt sie zu Rollen hinzu, die die Benutzer dann übernehmen können. IAM-Richtlinien definieren Berechtigungen unabhängig von der Methode, die zur Ausführung der Operation verwendet wird.

Identitätsbasierte Richtlinien

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität (Benutzer, Gruppe oder Rolle) anfügen können. Diese Richtlinien steuern, welche Aktionen Identitäten für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen

zum Erstellen identitätsbasierter Richtlinien finden Sie unter Definieren benutzerdefinierter IAM-Berechtigungen mit vom Kunden verwalteten Richtlinien im IAM-Benutzerhandbuch.

Identitätsbasierte Richtlinien können Inline-Richtlinien (direkt in eine einzelne Identität eingebettet) oder verwaltete Richtlinien (eigenständige Richtlinien, die mit mehreren Identitäten verbunden sind) sein. Informationen dazu, wie Sie zwischen verwalteten und Inline-Richtlinien wählen, finden Sie unter Choose between managed policies and inline policies im IAM-Benutzerhandbuch.

In der folgenden Tabelle werden die von Device Farm von AWS verwalteten Richtlinien beschrieben.

Änderungen	Beschreibung	Date
AWSDeviceFarmFullAccess	Bietet vollen Zugriff auf alle AWS Device Farm Farm-Operationen.	15. Juli 2015
AWSServiceRoleForDeviceFarmTestGrid	Ermöglicht Device Farm, in Ihrem Namen auf AWS-Ressourcen zuzugreifen.	20. Mai 2021

Weitere Richtlinientypen

AWS unterstützt zusätzliche Richtlinientypen, mit denen die maximalen Berechtigungen festgelegt werden können, die durch gängigere Richtlinientypen gewährt werden:

- Berechtigungsgrenzen – Eine Berechtigungsgrenze legt die maximalen Berechtigungen fest, die eine identitätsbasierte Richtlinie einer IAM-Entität erteilen kann. Weitere Informationen finden Sie unter Berechtigungsgrenzen für IAM-Entitäten im -IAM-Benutzerhandbuch.
- Richtlinien zur Dienstkontrolle (SCPs) — Geben Sie die maximalen Berechtigungen für eine Organisation oder Organisationseinheit in an AWS Organizations. Weitere Informationen finden Sie unter Service-Kontrollrichtlinien im AWS Organizations -Benutzerhandbuch.
- Richtlinien zur Ressourcenkontrolle (RCPs) — Legen Sie die maximal verfügbaren Berechtigungen für Ressourcen in Ihren Konten fest. Weitere Informationen finden Sie im AWS Organizations Benutzerhandbuch unter Richtlinien zur Ressourcenkontrolle (RCPs).
- Sitzungsrichtlinien – Sitzungsrichtlinien sind erweiterte Richtlinien, die als Parameter übergeben werden, wenn Sie eine temporäre Sitzung für eine Rolle oder einen Verbundbenutzer erstellen. Weitere Informationen finden Sie unter Sitzungsrichtlinien im IAM-Benutzerhandbuch.

Mehrere Richtlinientypen

Wenn für eine Anfrage mehrere Arten von Richtlinien gelten, sind die daraus resultierenden Berechtigungen schwieriger zu verstehen. Informationen darüber, wie AWS bestimmt wird, ob eine Anfrage zulässig ist, wenn mehrere Richtlinientypen betroffen sind, finden Sie unter Bewertungslogik für Richtlinien im IAM-Benutzerhandbuch.

Beispiele für identitätsbasierte Richtlinien von AWS Device Farm

Standardmäßig sind IAM-Benutzer und -Rollen nicht berechtigt, Device Farm Farm-Ressourcen zu erstellen oder zu ändern. Sie können auch keine Aufgaben mit der AWS-Managementkonsole AWS CLI, oder AWS API ausführen. Ein IAM-Administrator muss IAM-Richtlinien erstellen, die Benutzern und Rollen die Berechtigung zum Ausführen bestimmter API-Operationen für die angegebenen Ressourcen gewähren, die diese benötigen. Der Administrator muss diese Richtlinien anschließend den IAM-Benutzern oder -Gruppen anfügen, die diese Berechtigungen benötigen.

Informationen dazu, wie Sie unter Verwendung dieser beispielhaften JSON-Richtliniendokumente eine identitätsbasierte IAM-Richtlinie erstellen, finden Sie unter Erstellen von Richtlinien auf der JSON-Registerkarte im IAM-Benutzerhandbuch.

Themen

- Best Practices für Richtlinien
- Gewähren der Berechtigung zur Anzeige der eigenen Berechtigungen für Benutzer
- Zugreifen auf ein Device Farm Farm-Desktop-Browser-Testprojekt
- Anzeigen von Device Farm Farm-Desktop-Browser-Testprojekten auf der Grundlage von Tags

Best Practices für Richtlinien

Identitätsbasierte Richtlinien legen fest, ob jemand Device Farm Farm-Ressourcen in Ihrem Konto erstellen, darauf zugreifen oder sie löschen kann. Dies kann zusätzliche Kosten für Ihr verursachen AWS-Konto. Beachten Sie beim Erstellen oder Bearbeiten identitätsbasierter Richtlinien die folgenden Richtlinien und Empfehlungen:

- Erste Schritte mit AWS verwalteten Richtlinien und Umstellung auf Berechtigungen mit den geringsten Rechten — Verwenden Sie die AWS verwalteten Richtlinien, die Berechtigungen für viele gängige Anwendungsfälle gewähren, um damit zu beginnen, Ihren Benutzern und Workloads Berechtigungen zu gewähren. Sie sind in Ihrem verfügbar. AWS-Konto Wir empfehlen Ihnen,

die Berechtigungen weiter zu reduzieren, indem Sie vom AWS Kunden verwaltete Richtlinien definieren, die speziell auf Ihre Anwendungsfälle zugeschnitten sind. Weitere Informationen finden Sie unter Von AWS verwaltete Richtlinien oder Von AWS verwaltete Richtlinien für Auftragsfunktionen im IAM-Benutzerhandbuch.

- Anwendung von Berechtigungen mit den geringsten Rechten – Wenn Sie mit IAM-Richtlinien Berechtigungen festlegen, gewähren Sie nur die Berechtigungen, die für die Durchführung einer Aufgabe erforderlich sind. Sie tun dies, indem Sie die Aktionen definieren, die für bestimmte Ressourcen unter bestimmten Bedingungen durchgeführt werden können, auch bekannt als die geringsten Berechtigungen. Weitere Informationen zur Verwendung von IAM zum Anwenden von Berechtigungen finden Sie unter Richtlinien und Berechtigungen in IAM im IAM-Benutzerhandbuch.
- Verwenden von Bedingungen in IAM-Richtlinien zur weiteren Einschränkung des Zugriffs – Sie können Ihren Richtlinien eine Bedingung hinzufügen, um den Zugriff auf Aktionen und Ressourcen zu beschränken. Sie können beispielsweise eine Richtlinienbedingung schreiben, um festzulegen, dass alle Anforderungen mithilfe von SSL gesendet werden müssen. Sie können auch Bedingungen verwenden, um Zugriff auf Serviceaktionen zu gewähren, wenn diese für einen bestimmten Zweck verwendet werden AWS-Service, z. CloudFormation B. Weitere Informationen finden Sie unter IAM-JSON-Richtlinienelemente: Bedingung im IAM-Benutzerhandbuch.
- Verwenden von IAM Access Analyzer zur Validierung Ihrer IAM-Richtlinien, um sichere und funktionale Berechtigungen zu gewährleisten – IAM Access Analyzer validiert neue und vorhandene Richtlinien, damit die Richtlinien der IAM-Richtliniensprache (JSON) und den bewährten IAM-Methoden entsprechen. IAM Access Analyzer stellt mehr als 100 Richtlinienprüfungen und umsetzbare Empfehlungen zur Verfügung, damit Sie sichere und funktionale Richtlinien erstellen können. Weitere Informationen finden Sie unter Richtlinienvalidierung mit IAM Access Analyzer im IAM-Benutzerhandbuch.
- Multi-Faktor-Authentifizierung (MFA) erforderlich — Wenn Sie ein Szenario haben, das IAM-Benutzer oder einen Root-Benutzer in Ihrem System erfordert AWS-Konto, aktivieren Sie MFA für zusätzliche Sicherheit. Um MFA beim Aufrufen von API-Vorgängen anzufordern, fügen Sie Ihren Richtlinien MFA-Bedingungen hinzu. Weitere Informationen finden Sie unter Sicherer API-Zugriff mit MFA im IAM-Benutzerhandbuch.

Weitere Informationen zu bewährten Methoden in IAM finden Sie unter Best Practices für die Sicherheit in IAM im IAM-Benutzerhandbuch.

Gewähren der Berechtigung zur Anzeige der eigenen Berechtigungen für Benutzer

In diesem Beispiel wird gezeigt, wie Sie eine Richtlinie erstellen, die IAM-Benutzern die Berechtigung zum Anzeigen der eingebundenen Richtlinien und verwalteten Richtlinien gewährt, die ihrer Benutzeridentität angefügt sind. Diese Richtlinie umfasst Berechtigungen zum Ausführen dieser Aktion auf der Konsole oder programmgesteuert mithilfe der API oder. 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": "*"
        }
    ]
}
```

Zugreifen auf ein Device Farm Farm-Desktop-Browser-Testprojekt

In diesem Beispiel möchten Sie einem IAM-Benutzer in Ihrem AWS Konto Zugriff auf eines Ihrer Device Farm Farm-Desktop-Browser-Testprojekte gewähren. `arn:aws:devicefarm:us-west-2:111122223333:testgrid-project:123e4567-e89b-12d3-a456-426655441111` Sie möchten, dass Elemente im Konto angezeigt werden können, die mit dem Projekt zusammenhängen.

Zusätzlich zum `devicefarm:GetTestGridProject`-Endpunkt muss das Konto über die Endpunkte `devicefarm:ListTestGridSessions`, `devicefarm:GetTestGridSession`, `devicefarm:ListTestGridSessionActions` und `devicefarm:ListTestGridSessionArtifacts` verfügen.

Wenn Sie CI-Systeme verwenden, sollten Sie für jeden CI Runner eindeutige Anmeldeinformationen festlegen. Beispielsweise ist es unwahrscheinlich, dass ein CI-System mehr Berechtigungen als `devicefarm:ScheduleRun` oder `devicefarm:CreateUpload` benötigt. Die folgende IAM-Richtlinie beschreibt eine Mindestrichtlinie, die es einem CI-Runner ermöglicht, einen Test eines neuen nativen Device Farm Farm-App-Tests zu starten, indem er einen Upload erstellt und damit einen Testlauf plant:

Anzeigen von Device Farm Farm-Desktop-Browser-Testprojekten auf der Grundlage von Tags

Sie können Bedingungen in Ihrer identitätsbasierten Richtlinie verwenden, um den Zugriff auf Device Farm Farm-Ressourcen anhand von Tags zu steuern. In diesem Beispiel wird gezeigt, wie Sie eine Richtlinie erstellen, um Projekte und Sitzungen anzuzeigen. Die Berechtigung wird erteilt, wenn das `0wner`-Tag der angeforderten Ressource mit dem Benutzernamen des anfordernden Kontos übereinstimmt.

Sie können diese Richtlinie den IAM-Benutzern in Ihrem Konto anfügen. Wenn ein benannter Benutzer `richard-roe` versucht, ein Device Farm-Projekt oder eine Device Farm-Sitzung aufzurufen, muss das Projekt mit `0wner=richard-roe` oder markiert werden`owner=richard-roe`. Andernfalls wird dem Benutzer der Zugriff verweigert. Der Tag-Schlüssel `0wner` der Bedingung stimmt sowohl mit `0wner` als auch mit `owner` überein, da die Namen von Bedingungsschlüsseln nicht zwischen Groß- und Kleinschreibung unterscheiden. Weitere Informationen finden Sie unter IAM-JSON-Richtlinienelemente: Bedingung im IAM-Benutzerhandbuch.

Fehlerbehebung bei Identität und Zugriff auf AWS Device Farm

Verwenden Sie die folgenden Informationen, um häufig auftretende Probleme zu diagnostizieren und zu beheben, die bei der Arbeit mit Device Farm und IAM auftreten können.

Ich bin nicht berechtigt, eine Aktion in Device Farm durchzuführen

Wenn Sie eine Fehlermeldung erhalten AWS-Managementkonsole , die besagt, dass Sie nicht berechtigt sind, eine Aktion auszuführen, müssen Sie sich an Ihren Administrator wenden, um Unterstützung zu erhalten. Ihr Administrator ist die Person, die Ihnen Ihren Benutzernamen und Ihr Passwort bereitgestellt hat.

Der folgende Beispielfehler tritt auf, wenn der IAM-Benutzer,`mateojackson`, versucht, die Konsole zu verwenden, um Details zu einem Lauf anzuzeigen, aber nicht über die `devicefarm:GetRun` entsprechenden Berechtigungen verfügt.

```
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
```

In diesem Fall bittet Mateo den Administrator, die Richtlinien zu aktualisieren, um ihm den Zugriff zu `devicefarm:GetRun` auf der Ressource `arn:aws:devicefarm:us-west-2:123456789101:run:123e4567-e89b-12d3-a456-426655440000/123e4567-e89b-12d3-a456-426655441111` über die Aktion `devicefarm:GetRun` zu ermöglichen.

Ich bin nicht berechtigt, IAM auszuführen: PassRole

Wenn Sie eine Fehlermeldung erhalten, dass Sie nicht berechtigt sind, die `iam:PassRole` Aktion auszuführen, müssen Ihre Richtlinien aktualisiert werden, damit Sie eine Rolle an Device Farm übergeben können.

Einige AWS-Services ermöglichen es Ihnen, eine bestehende Rolle an diesen Dienst zu übergeben, anstatt eine neue Servicerolle oder eine dienstverknüpfte Rolle zu erstellen. Hierzu benötigen Sie Berechtigungen für die Übergabe der Rolle an den Dienst.

Der folgende Beispielfehler tritt auf, wenn ein IAM-Benutzer mit dem Namen `marymajor` versucht, die Konsole zu verwenden, um eine Aktion in Device Farm auszuführen. Die Aktion erfordert jedoch, dass der Service über Berechtigungen verfügt, die durch eine Servicerolle gewährt werden. Mary besitzt keine Berechtigungen für die Übergabe der Rolle an den Dienst.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In diesem Fall müssen die Richtlinien von Mary aktualisiert werden, um die Aktion `iam:PassRole` ausführen zu können.

Wenn Sie Hilfe benötigen, wenden Sie sich an Ihren AWS Administrator. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen odzur Verfügung gestellt.

Ich möchte meine Zugriffsschlüssel anzeigen

Nachdem Sie Ihre IAM-Benutzerzugriffsschlüssel erstellt haben, können Sie Ihre Zugriffsschlüssel-ID jederzeit anzeigen. Sie können Ihren geheimen Zugriffsschlüssel jedoch nicht erneut anzeigen. Wenn Sie den geheimen Zugriffsschlüssel verlieren, müssen Sie ein neues Zugriffsschlüsselpaar erstellen.

Zugriffsschlüssel bestehen aus zwei Teilen: einer Zugriffsschlüssel-ID (z. B. `AKIAIOSFODNN7EXAMPLE`) und einem geheimen Zugriffsschlüssel (z. B. `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Ähnlich wie bei Benutzernamen und Passwörtern müssen Sie die Zugriffsschlüssel-ID und den geheimen Zugriffsschlüssel zusammen verwenden, um Ihre Anforderungen zu authentifizieren. Verwalten Sie Ihre Zugriffsschlüssel so sicher wie Ihren Benutzernamen und Ihr Passwort.

> **Important**
>
> Geben Sie Ihre Zugriffsschlüssel nicht an Dritte weiter, auch nicht für die Suche nach Ihrer kanonischen Benutzer-ID. Auf diese Weise können Sie jemandem dauerhaften Zugriff auf Ihre gewähren AWS-Konto.

Während der Erstellung eines Zugriffsschlüsselpaars werden Sie aufgefordert, die Zugriffsschlüssel-ID und den geheimen Zugriffsschlüssel an einem sicheren Speicherort zu speichern. Der geheime Zugriffsschlüssel ist nur zu dem Zeitpunkt verfügbar, an dem Sie ihn erstellen. Wenn Sie Ihren geheimen Zugriffsschlüssel verlieren, müssen Sie Ihrem IAM-Benutzer neue Zugriffsschlüssel hinzufügen. Sie können maximal zwei Zugriffsschlüssel besitzen. Wenn Sie bereits zwei Zugriffschlüssel besitzen, müssen Sie ein Schlüsselpaar löschen, bevor Sie ein neues erstellen. Anweisungen hierfür finden Sie unter Verwalten von Zugriffsschlüsseln im IAM-Benutzerhandbuch.

Ich bin Administrator und möchte anderen den Zugriff auf Device Farm ermöglichen

Um anderen den Zugriff auf Device Farm zu ermöglichen, müssen Sie den Personen oder Anwendungen, die Zugriff benötigen, die entsprechenden Berechtigungen erteilen. Wenn Sie Benutzer und Anwendungen verwalten, weisen Sie Benutzern oder Gruppen Berechtigungssätze zu, um deren Zugriffsebene zu definieren. AWS IAM Identity Center Mit Berechtigungssätzen werden automatisch IAM-Richtlinien erstellt und den IAM-Rollen zugewiesen, die der Person oder Anwendung zugeordnet sind. Weitere Informationen finden Sie im AWS IAM Identity Center Benutzerhandbuch unter Berechtigungssätze.

Wenn Sie IAM Identity Center nicht verwenden, müssen Sie IAM-Entitäten (Benutzer oder Rollen) für die Personen oder Anwendungen erstellen, die Zugriff benötigen. Anschließend müssen Sie der Entität eine Richtlinie hinzufügen, die ihr die richtigen Berechtigungen in Device Farm gewährt. Nachdem die Berechtigungen erteilt wurden, geben Sie die Anmeldeinformationen an den Benutzer oder Anwendungsentwickler weiter. Sie werden diese Anmeldeinformationen für den Zugriff verwenden AWS. Weitere Informationen zum Erstellen von IAM-Benutzern, -Gruppen, -Richtlinien und -Berechtigungen finden Sie im IAM-Benutzerhandbuch unter IAM-Identitäten sowie Richtlinien und Berechtigungen in IAM.

Ich möchte Personen außerhalb meines AWS Kontos den Zugriff auf meine Device Farm Farm-Ressourcen ermöglichen

Sie können eine Rolle erstellen, mit der Benutzer in anderen Konten oder Personen außerhalb Ihrer Organisation auf Ihre Ressourcen zugreifen können. Sie können festlegen, wem die Übernahme der Rolle anvertraut wird. Für Dienste, die ressourcenbasierte Richtlinien oder Zugriffskontrolllisten (ACLs) unterstützen, können Sie diese Richtlinien verwenden, um Personen Zugriff auf Ihre Ressourcen zu gewähren.

Weitere Informationen dazu finden Sie hier:

- Informationen darüber, ob Device Farm diese Funktionen unterstützt, finden Sie unterSo funktioniert AWS Device Farm mit IAM.
- Informationen dazu, wie Sie Zugriff auf Ihre Ressourcen gewähren können, AWS-Konten die Ihnen gehören, finden Sie im IAM-Benutzerhandbuch unter Gewähren des Zugriffs für einen IAM-Benutzer in einem anderen AWS-Konto , den Sie besitzen.
- Informationen dazu, wie Sie Dritten Zugriff auf Ihre Ressourcen gewähren können AWS-Konten, finden Sie AWS-Konten im IAM-Benutzerhandbuch unter Gewähren des Zugriffs für Dritte.

- Informationen dazu, wie Sie über einen Identitätsverbund Zugriff gewähren, finden Sie unter Gewähren von Zugriff für extern authentifizierte Benutzer (Identitätsverbund) im IAM-Benutzerhandbuch.
- Informationen zum Unterschied zwischen der Verwendung von Rollen und ressourcenbasierten Richtlinien für den kontoübergreifenden Zugriff finden Sie unter Kontoübergreifender Ressourcenzugriff in IAM im IAM-Benutzerhandbuch.

Konformitätsvalidierung für AWS Device Farm

Externe Prüfer bewerten die Sicherheit und Einhaltung von Vorschriften im AWS Device Farm Rahmen mehrerer AWS Compliance-Programme. Dazu gehören SOC, PCI, FedRAMP, HIPAA und andere. AWS Device Farm fällt nicht in den Geltungsbereich irgendwelcher Compliance-Programme. AWS

Eine Liste der AWS Services im Rahmen bestimmter Compliance-Programme finden Sie unter AWS-Services in Umfang nach Compliance-Programm . Allgemeine Informationen finden Sie unter AWS Compliance-Programme AWS .

Sie können Prüfberichte von Drittanbietern unter herunterladen AWS Artifact. Weitere Informationen finden Sie unter Herunterladen von Berichten in AWS Artifact.

Ihre Compliance-Verantwortung bei der Nutzung von Device Farm hängt von der Sensibilität Ihrer Daten, den Compliance-Zielen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften ab. AWS stellt die folgenden Ressourcen zur Verfügung, die Sie bei der Einhaltung der Vorschriften unterstützen:

- Schnellstartanleitungen für Sicherheit und Compliance – In diesen Bereitstellungsleitfäden werden architektonische Überlegungen erörtert und Schritte für die Bereitstellung von sicherheits- und konformitätsorientierten Basisumgebungen auf AWS angegeben.
- AWS Ressourcen zur AWS von Vorschriften — Diese Sammlung von Arbeitsmappen und Leitfäden kann auf Ihre Branche und Ihren Standort zutreffen.
- Die Bewertung von Ressourcen anhand von Regeln im AWS Config Entwicklerhandbuch — AWS Config bewertet, wie gut Ihre Ressourcenkonfigurationen den internen Praktiken, Branchenrichtlinien und Vorschriften entsprechen.
- AWS Security Hub CSPM— Dieser AWS Service bietet einen umfassenden Überblick über Ihren Sicherheitsstatus, sodass Sie überprüfen können AWS , ob Sie die Sicherheitsstandards und Best Practices der Branche einhalten.

Datenschutz in AWS Device Farm

Das AWS Modell der mit gilt für den Datenschutz in AWS Device Farm (Device Farm). Wie in diesem Modell beschrieben, AWS ist es verantwortlich für den Schutz der globalen Infrastruktur, auf der die gesamte Infrastruktur läuft AWS Cloud. Sie sind dafür verantwortlich, die Kontrolle über Ihre in dieser Infrastruktur gehosteten Inhalte zu behalten. Sie sind auch für die Sicherheitskonfiguration und die Verwaltungsaufgaben für die von Ihnen verwendeten AWS-Services verantwortlich. Weitere Informationen zum Datenschutz finden Sie unter Häufig gestellte Fragen zum Datenschutz. Informationen zum Datenschutz in Europa finden Sie im Blog-Beitrag AWS -Modell der geteilten Verantwortung und in der DSGVO im AWS -Sicherheitsblog.

Aus Datenschutzgründen empfehlen wir, dass Sie AWS-Konto Anmeldeinformationen schützen und einzelne Benutzer mit AWS IAM Identity Center oder AWS Identity and Access Management (IAM) einrichten. So erhält jeder Benutzer nur die Berechtigungen, die zum Durchführen seiner Aufgaben erforderlich sind. Außerdem empfehlen wir, die Daten mit folgenden Methoden schützen:

- Verwenden Sie für jedes Konto die Multi-Faktor-Authentifizierung (MFA).
- Wird verwendet SSL/TLS , um mit AWS Ressourcen zu kommunizieren. Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Richten Sie die API und die Protokollierung von Benutzeraktivitäten mit ein AWS CloudTrail. Informationen zur Verwendung von CloudTrail Pfaden zur Erfassung von AWS Aktivitäten finden Sie unter Arbeiten mit CloudTrail Pfaden im AWS CloudTrail Benutzerhandbuch.
- Verwenden Sie AWS Verschlüsselungslösungen zusammen mit allen darin enthaltenen Standardsicherheitskontrollen AWS-Services.
- Verwenden Sie erweiterte verwaltete Sicherheitsservices wie Amazon Macie, die dabei helfen, in Amazon S3 gespeicherte persönliche Daten zu erkennen und zu schützen.
- Wenn Sie für den Zugriff AWS über eine Befehlszeilenschnittstelle oder eine API FIPS 140-3-validierte kryptografische Module benötigen, verwenden Sie einen FIPS-Endpunkt. Weitere Informationen über verfügbare FIPS-Endpunkte finden Sie unter Federal Information Processing Standard (FIPS) 140-3.

Wir empfehlen dringend, in Freitextfeldern, z. B. im Feld Name, keine vertraulichen oder sensiblen Informationen wie die E-Mail-Adressen Ihrer Kunden einzugeben. Dies gilt auch, wenn Sie mit Device Farm oder anderen Geräten arbeiten und die Konsole, die API oder AWS-Services verwenden AWS SDKs. AWS CLI Alle Daten, die Sie in Tags oder Freitextfelder eingeben, die für Namen verwendet werden, können für Abrechnungs- oder Diagnoseprotokolle verwendet werden. Wenn Sie eine URL

für einen externen Server bereitstellen, empfehlen wir dringend, keine Anmeldeinformationen zur Validierung Ihrer Anforderung an den betreffenden Server in die URL einzuschließen.

Verschlüsselung während der Übertragung

Die Device Farm Farm-Endpunkte unterstützen nur signiertes HTTPS (SSL/TLS) requests except where otherwise noted. All content retrieved from or placed in Amazon S3 through upload URLs is encrypted using SSL/TLS. Weitere Informationen darüber, wie HTTPS-Anfragen angemeldet werden AWS, finden Sie in der AWS Allgemeinen Referenz unter Signieren von AWS API-Anfragen.

Sie sind verantwortlich für die Verschlüsselung und das Sichern der Kommunikationen von Ihren getesteten Anwendungen und allen Anwendungen, die bei der Ausführung von Tests auf dem Gerät installiert wurden.

Verschlüsselung im Ruhezustand

Die Desktop-Browser-Testfunktion von Device Farm unterstützt die Verschlüsselung im Ruhezustand für Artefakte, die während der Tests generiert wurden.

Die Testdaten von Device Farm auf physischen Mobilgeräten werden im Ruhezustand nicht verschlüsselt.

Datenaufbewahrung

Daten in Device Farm werden für eine begrenzte Zeit aufbewahrt. Nach Ablauf der Aufbewahrungsfrist werden die Daten aus dem Backup-Speicher von Device Farm entfernt.

Art des Inhalts	Aufbewahrungszeitraum (Tage)	Aufbewahrungszeitraum für Metadaten (Tage)
Hochgeladene Anwendungen	30	30
Hochgeladene Testpakete	30	30
Protokolle	400	400
Videoaufnahmen und andere Artefakte	400	400

Es liegt in Ihrer Verantwortung, Inhalte zu archivieren, die Sie länger aufbewahren möchten.

Datenverwaltung

Daten in Device Farm werden je nachdem, welche Funktionen verwendet werden, unterschiedlich verwaltet. In diesem Abschnitt wird erklärt, wie Daten während und nach der Verwendung von Device Farm verwaltet werden.

Testen von Desktop-Browsern

Instances, die während Selenium-Sitzungen verwendet werden, werden nicht gespeichert. Alle Daten, die durch Browserinteraktionen generiert werden, werden beim Ende der Sitzung verworfen.

Diese Funktion unterstützt derzeit die Verschlüsselung im Ruhezustand für Artefakte, die während des Tests generiert wurden.

Testen physischer Geräte

In den folgenden Abschnitten finden Sie Informationen zu den Schritten, AWS die zum Bereinigen oder Löschen von Geräten ergriffen werden, nachdem Sie Device Farm verwendet haben.

Die Testdaten von Device Farm auf physischen Mobilgeräten sind im Ruhezustand nicht verschlüsselt.

Öffentliche Geräteflotten

Nach Abschluss der Testausführung führt Device Farm eine Reihe von Bereinigungsaufgaben auf jedem Gerät in der öffentlichen Geräteflotte durch, einschließlich der Deinstallation Ihrer App. Wenn wir die Deinstallation Ihrer Anwendung oder einen der anderen Bereinigungsschritt nicht überprüfen können, wird das Gerät vor der Wiederinbetriebnahme zurückgesetzt.

> **Note**
>
> In einigen Fällen ist es möglich, dass Daten zwischen den Sitzungen bestehen bleiben, insbesondere wenn Sie das Gerätesystem außerhalb des Kontextes Ihrer App verwenden. Aus diesem Grund und weil Device Farm Videos und Protokolle von Aktivitäten erfasst, die während Ihrer Nutzung der einzelnen Geräte stattfinden, empfehlen wir, dass Sie während Ihrer automatisierten Test- und Fernzugriffssitzungen keine vertraulichen Informationen (z. B. Google-Konto oder Apple-ID), persönliche Daten und andere sicherheitsrelevante Daten eingeben.

Private Geräte

Nach dem Ablauf oder der Beendigung Ihres Vertrags über ein privates Gerät wird das Gerät außer Betrieb genommen und in Übereinstimmung mit AWS-Richtlinien für die Löschung sicher gelöscht. Weitere Informationen finden Sie unter Private Geräte in AWS Device Farm.

Schlüsselverwaltung

Derzeit bietet Device Farm kein externes Schlüsselmanagement für die Verschlüsselung von Daten im Ruhezustand oder bei der Übertragung.

Richtlinie für den Datenverkehr zwischen Netzwerken

Device Farm kann nur für private Geräte so konfiguriert werden, dass Amazon VPC-Endpunkte verwendet werden, um eine Verbindung zu Ihren Ressourcen herzustellen. AWS Der Zugriff auf jegliche nichtöffentliche AWS Infrastruktur, die mit Ihrem Konto verknüpft ist (z. B. EC2 Amazon-Instances ohne öffentliche IP-Adresse), muss einen Amazon VPC-Endpunkt verwenden. Unabhängig von der VPC-Endpunktkonfiguration isoliert Device Farm Ihren Datenverkehr von anderen Benutzern im gesamten Device Farm Farm-Netzwerk.

Es kann nicht garantiert werden, dass Ihre Verbindungen außerhalb des AWS Netzwerks gesichert oder sicher sind, und es liegt in Ihrer Verantwortung, alle Internetverbindungen, die Ihre Anwendungen herstellen, zu sichern.

Resilienz in AWS Device Farm

Die AWS globale Infrastruktur basiert auf AWS Regionen und Availability Zones. AWS Regionen bieten mehrere physisch getrennte und isolierte Availability Zones, die über Netzwerke mit niedriger Latenz, hohem Durchsatz und hoher Redundanz miteinander verbunden sind. Mithilfe von Availability Zones können Sie Anwendungen und Datenbanken erstellen und ausführen, die automatisch Failover zwischen Zonen ausführen, ohne dass es zu Unterbrechungen kommt. Availability Zones sind besser verfügbar, fehlertoleranter und skalierbarer als herkömmliche Infrastrukturen mit einem oder mehreren Rechenzentren.

Weitere Informationen zu AWS Regionen und Availability Zones finden Sie unter AWS Globale Infrastruktur.

Da Device Farm nur in der `us-west-2` Region verfügbar ist, empfehlen wir dringend, Sicherungs- und Wiederherstellungsprozesse zu implementieren. Device Farm sollte nicht die einzige Quelle für hochgeladene Inhalte sein.

Device Farm garantiert nicht die Verfügbarkeit öffentlicher Geräte. Diese Geräte werden aufgrund einer Vielzahl von Faktoren wie etwa Ausfallrate und Quarantänestatus in den öffentlichen Gerätepool eingebunden oder daraus entfernt. Wir empfehlen nicht, sich von der Verfügbarkeit eines Geräts im öffentlichen Gerätepool abhängig zu machen.

Infrastruktursicherheit in AWS Device Farm

Als verwalteter Dienst AWS Device Farm wird er durch die AWS globale Netzwerksicherheit geschützt. Informationen zu AWS Sicherheitsdiensten und zum AWS Schutz der Infrastruktur finden Sie unter AWS Cloud-Sicherheit. Informationen zum Entwerfen Ihrer AWS Umgebung unter Verwendung der bewährten Methoden für die Infrastruktursicherheit finden Sie unter Infrastructure Protection in Security Pillar AWS Well-Architected Framework.

Sie verwenden AWS veröffentlichte API-Aufrufe, um über das Netzwerk auf Device Farm zuzugreifen. Kunden müssen Folgendes unterstützen:

- Transport Layer Security (TLS). Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Verschlüsselungs-Suiten mit Perfect Forward Secrecy (PFS) wie DHE (Ephemeral Diffie-Hellman) oder ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Die meisten modernen Systeme wie Java 7 und höher unterstützen diese Modi.

Außerdem müssen Anforderungen mit einer Zugriffsschlüssel-ID und einem geheimen Zugriffsschlüssel signiert sein, der einem IAM-Prinzipal zugeordnet ist. Alternativ können Sie mit AWS -Security-Token-Service (AWS STS) temporäre Sicherheitsanmeldeinformationen erstellen, um die Anforderungen zu signieren.

Sicherheit der Infrastruktur für Tests physischer Geräte

Geräte werden während des Tests physisch getrennt. Die Netzwerkisolierung verhindert die geräteübergreifende Kommunikation über drahtlose Netzwerke.

Öffentliche Geräte werden gemeinsam genutzt, und Device Farm bemüht sich nach besten Kräften, die Geräte langfristig zu schützen. Bestimmte Aktionen, z. B. Versuche, vollständige Administratorrechte auf einem Gerät zu erwerben (eine Praxis, die als Rooting oder Jailbreak bezeichnet wird), führen dazu, dass öffentliche Geräte isoliert werden. Sie werden automatisch aus dem öffentlichen Pool entfernt und manuell überprüft.

Auf private Geräte kann nur über AWS Konten zugegriffen werden, die ausdrücklich dazu autorisiert sind. Device Farm isoliert diese Geräte physisch von anderen Geräten und hält sie in einem separaten Netzwerk.

Auf privat verwalteten Geräten können Tests so konfiguriert werden, dass sie einen Amazon VPC-Endpunkt verwenden, um Verbindungen in und aus Ihrem AWS Konto zu sichern.

Sicherheit der Infrastruktur für das Testen von Desktop-Browsern

Wenn Sie die Funktion für Desktop-Browsertests verwenden, werden alle Testsitzungen voneinander getrennt. Selenium-Instanzen können ohne eine externe Zwischenpartei nicht miteinander kommunizieren. AWS

Der gesamte Datenverkehr zu WebDriver Selenium-Controllern muss über den mit generierten HTTPS-Endpunkt erfolgen. `createTestGridUrl`

Sie sind dafür verantwortlich, dass jede Device Farm Farm-Testinstanz sicheren Zugriff auf die von ihr getesteten Ressourcen hat. Standardmäßig haben die Desktop-Browser-Testinstanzen von Device Farm Zugriff auf das öffentliche Internet. Wenn Sie Ihre Instance an eine VPC anhängen, verhält sie sich wie jede andere EC2-Instance, wobei der Zugriff auf Ressourcen durch die Konfiguration der VPC und die zugehörigen Netzwerkkomponenten bestimmt wird. AWS bietet Sicherheitsgruppen und Netzwerk-Zugriffskontrolllisten (ACLs), um die Sicherheit in Ihrer VPC zu erhöhen. Sicherheitsgruppen kontrollieren den ein- und ausgehenden Datenverkehr für Ihre Ressourcen, und das Netzwerk ACLs steuert den ein- und ausgehenden Datenverkehr für Ihre Subnetze. Sicherheitsgruppen bieten ausreichend Zugriffskontrolle für die meisten Subnetze. Sie können das Netzwerk verwenden ACLs , wenn Sie eine zusätzliche Sicherheitsebene für Ihre VPC wünschen. Allgemeine Richtlinien zu bewährten Sicherheitsmethoden bei der Nutzung von Amazon VPCs finden Sie unter Bewährte Sicherheitsmethoden für Ihre VPC im Amazon Virtual Private Cloud Cloud-Benutzerhandbuch.

Analyse und Verwaltung von Konfigurationsschwachstellen in Device Farm

Device Farm ermöglicht es Ihnen, Software auszuführen, die nicht aktiv vom Anbieter gewartet oder gepatcht wird, z. B. vom Betriebssystemanbieter, Hardwareanbieter oder Telefonanbieter. Device Farm bemüht sich nach besten Kräften, die Software auf dem neuesten Stand zu halten, garantiert jedoch nicht, dass eine bestimmte Version der Software auf einem physischen Gerät auf dem

neuesten Stand ist, sodass potenziell anfällige Software so konzipiert ist, dass potenziell anfällige Software verwendet werden kann.

Wenn beispielsweise ein Test auf einem Gerät mit Android 4.4.2 durchgeführt wird, garantiert Device Farm nicht, dass das Gerät gegen die Sicherheitsanfälligkeit in Android gepatcht ist, die als bekannt ist. StageFright Es ist Sache des Herstellers (und manchmal auch des Anbieters) des Geräts, Sicherheitsupdates für Geräte bereitzustellen. Bei einer bösartigen Anwendung, die diese Schwachstelle ausnutzt, ist nicht garantiert, dass sie von unserer automatisierten Quarantäne erfasst wird.

Private Geräte werden gemäß Ihrer Vereinbarung mit verwaltet. AWS

Device Farm bemüht sich nach besten Kräften, Kundenanwendungen vor Aktionen wie Rooting oder Jailbreak zu schützen. Device Farm entfernt Geräte, die unter Quarantäne gestellt wurden, aus dem öffentlichen Pool, bis sie manuell überprüft wurden.

Sie sind dafür verantwortlich, alle Bibliotheken oder Versionen von Software, die Sie in Ihren Tests verwenden, wie Python-Wheels und Ruby-Gems, auf dem neuesten Stand zu halten. Device Farm empfiehlt, dass Sie Ihre Testbibliotheken aktualisieren.

Diese Ressourcen können dazu beitragen, Ihre Testabhängigkeiten auf dem neuesten Stand zu halten:

- Informationen zur Sicherung von Ruby-Gems finden Sie auf der RubyGems Website unter Sicherheitspraktiken.
- Informationen über das Sicherheitspaket, das von Pipenv verwendet und von der Python Packaging Authority unterstützt wird, um Ihr Abhängigkeitsdiagramm nach bekannten Sicherheitslücken zu durchsuchen, finden Sie unter Erkennung von Sicherheitslücken auf. GitHub
- Informationen zum Maven-Abhängigkeitsprüfer des Open Web Application Security Project (OWASP) finden Sie unter OWASP auf der OWASP-Website. DependencyCheck

Denken Sie daran, dass selbst wenn ein automatisiertes System keine bekannten Sicherheitsprobleme meldet, dies nicht bedeutet, dass keine vorhanden sein können. Lassen Sie immer Sorgfalt walten, wenn Sie Bibliotheken oder Tools von Drittanbietern verwenden, und überprüfen Sie kryptografische Signaturen, sofern möglich.

Reaktion auf Vorfälle in Device Farm

Device Farm überwacht Geräte kontinuierlich auf Verhaltensweisen, die auf Sicherheitsprobleme hinweisen könnten. Wenn ein Kunde auf einen Fall aufmerksam gemacht AWS wird, in dem Kundendaten wie Testergebnisse oder Dateien, die auf ein öffentliches Gerät geschrieben wurden, für einen anderen Kunden zugänglich sind, AWS kontaktiert er die betroffenen Kunden gemäß den standardmäßigen Richtlinien für Warnung und Berichterstattung bei Vorfällen, die in allen AWS Diensten verwendet werden.

Protokollierung und Überwachung in Device Farm

Dieser Service unterstützt AWS CloudTrail. Dabei handelt es sich um einen Service, der AWS Anrufe für Sie aufzeichnet AWS-Konto und Protokolldateien an einen Amazon S3 S3-Bucket übermittelt. Anhand der von gesammelten Informationen können Sie feststellen CloudTrail, an welche Anfragen erfolgreich gestellt wurden AWS-Services, wer die Anfrage gestellt hat, wann sie gestellt wurde usw. Weitere Informationen darüber CloudTrail, wie Sie es einschalten und wie Sie Ihre Protokolldateien finden, finden Sie im AWS CloudTrail Benutzerhandbuch.

Informationen zur Verwendung CloudTrail mit Device Farm finden Sie unterProtokollieren von API-Aufrufen von AWS Device Farm mit AWS CloudTrail.

Bewährte Sicherheitsmethoden für Device Farm

Device Farm bietet eine Reihe von Sicherheitsfunktionen, die Sie bei der Entwicklung und Implementierung Ihrer eigenen Sicherheitsrichtlinien berücksichtigen sollten. Die folgenden bewährten Methoden sind allgemeine Richtlinien und keine vollständige Sicherheitslösung. Da diese bewährten Methoden für Ihre Umgebung möglicherweise nicht angemessen oder ausreichend sind, sollten Sie sie als hilfreiche Überlegungen und nicht als bindend ansehen.

- Gewähren Sie jedem System der kontinuierlichen Integration (Continuous Integration System, CI) unter IAM die geringstmöglichen Berechtigungen. Erwägen Sie, temporäre Anmeldeinformationen für jeden CI-Systemtest zu verwenden, so dass ein CI-System auch bei einer Kompromittierung keine falsche Anforderungen stellen kann. Weitere Informationen zu temporären Anmeldeinformationen finden Sie im IAM-Benutzerhandbuch.
- Verwenden Sie adb-Befehle in einer benutzerdefinierten Testumgebung, um von Ihrer Anwendung erstellte Inhalte zu bereinigen. Weitere Informationen über benutzerdefinierten Testumgebungen finden Sie unter Benutzerdefinierte Testumgebungen.

Grenzwerte in der AWS-Gerätefarm

Themen

Service Limits

- Es gibt keine Einschränkungen hinsichtlich der Anzahl an Geräten, die in einem Testlauf berücksichtigt werden können. Die maximale Anzahl von Geräten, die Device Farm während eines Testlaufs gleichzeitig testet, beträgt jedoch fünf. Diese Anzahl kann auf Anfrage erhöht und von Fall zu Fall vom Serviceteam bewertet werden.
- Es gibt keine Einschränkungen in Bezug auf die Anzahl an Testläufen, die Sie planen können. Beachten Sie, dass sie nur bis zu 24 Stunden in der Warteschlange bleiben können.
- Die Dauer einer Fernzugriffssitzung ist auf 150 Minuten begrenzt.
- Die Dauer eines automatisierten Testlaufs ist auf 150 Minuten begrenzt
- Die maximale Anzahl laufender Aufträge, einschließlich ausstehender Aufträge in der Warteschlange, in Ihrem Konto beträgt 250. Dies ist ein Soft-Limit.
- Die Anzahl der Geräte, die Sie in einen Testlauf einbeziehen können, ist unbegrenzt. Die Anzahl der Geräte (Jobs), die Ihre Tests zu einem bestimmten Zeitpunkt parallel ausführen können, entspricht Ihrer Parallelität auf Kontoebene. Die Standardparallelität auf Kontoebene für die kostenpflichtige Nutzung in Device Farm ist fünf.
- Das Limit für zeitgesteuerte Parallelität kann auf Anfrage je nach Anwendungsfall bis zu einem bestimmten Schwellenwert erhöht werden. Die standardmäßige Parallelität auf Kontoebene für die Nutzung ohne Zeitlimit entspricht der Anzahl der Slots, die Sie für diese Plattform abonniert haben.

  Weitere Informationen zu den standardmäßigen Grenzwerten für gemessene Parallelität oder zu den Quoten im Allgemeinen finden Sie auf der Seite Kontingente.
- Ein Automatisierungslauf, der keine benutzerdefinierte Testumgebung verwendet, kann nur bis zu 250 einzelne Testfälle enthalten. Andernfalls kann der Lauf übersprungen werden.

Dateibeschränkungen

- Die maximale Größe einer App-Datei, die Sie hochladen können, beträgt 4 GB. Beachten Sie, dass wir derzeit keine Dateien im AAB-Format für Android akzeptieren.
- Die maximale Größe des automatisch generierten Videos von Device Farm während Ihres Testlaufs beträgt 1 GB. Bei Videos, die diese Größe überschreiten, wird der gesamte verbleibende Videoinhalt gekürzt. Kunden können weiterhin ihre eigene Videoaufzeichnungslösung verwenden, sofern vorhanden, und diese außerhalb des verwalteten Speichers von Device Farm speichern.
- Die maximale Größe des automatisch generierten Geräteprotokolls von Device Farm (Logcat auf Android oder Syslog auf iOS) während Ihres Testlaufs beträgt 1 GB. Bei Protokollen, die diese Größe überschreiten, werden alle verbleibenden Protokolle gekürzt. Bei Protokollen, die größer als 1 GB sind, können Kunden diese Protokolle außerhalb des verwalteten Speichers von Device Farm speichern.
- Die maximale Gesamtgröße der Kundenartefakte im benutzerdefinierten Umgebungsmodus von Device Farm beträgt 1 GB. Wenn diese Größe von Ihren Artefakten überschritten wird, ist keines der Artefakte verfügbar.
- Wenn die Gesamtgröße aller während eines Testlaufs generierten Artefakte 4 GB überschreitet, werden möglicherweise einige Artefakte gelöscht (einschließlich Video, Geräteprotokolle und Kundenartefakte).

API-Limits

- Device Farm folgt einem Token-Bucket-Algorithmus, um die API-Aufrufraten zu drosseln. Stellen Sie sich zum Beispiel vor, Sie erstellen einen Bucket, der Tokens enthält. Jedes Token steht für eine Transaktion, und ein API-Aufruf verbraucht ein Token. Token werden dem Bucket mit einer festen Rate hinzugefügt (z. B. 10 Token pro Sekunde), und der Bucket hat eine maximale Kapazität (z. B. 100 Token). Wenn eine Anfrage oder ein Paket eintrifft, muss es ein Token aus dem Bucket anfordern, damit es verarbeitet werden kann. Wenn genügend Token vorhanden sind, wird die Anfrage zugelassen und die Token werden entfernt. Wenn nicht genügend Token vorhanden sind, wird die Anfrage je nach Implementierung entweder verzögert oder gelöscht.

  In Device Farm wird der Algorithmus so implementiert:

  - Bei den Burst-API-Anfragen handelt es sich um die maximale Anzahl von Anfragen, auf die der Dienst für eine bestimmte API in einer bestimmten Kundenkonto-ID antworten kann. Mit anderen

Worten, dies ist die Kapazität des Buckets. Sie können die API so oft aufrufen, wie Token im Bucket verbleiben, und jede Anfrage verbraucht ein Token.

- Die Transactions-per-second (TPS-) Rate ist die Mindestrate, mit der Ihre API-Anfragen ausgeführt werden können. Mit anderen Worten, dies ist die Geschwindigkeit, mit der der Bucket pro Sekunde mit Tokens aufgefüllt wird. Wenn eine API beispielsweise eine Burst-Zahl von zehn, aber einen TPS-Wert von eins hat, könnten Sie sie sofort zehnmal aufrufen. Der Bucket würde jedoch nur Token mit einer Geschwindigkeit von einem Token pro Sekunde zurückgewinnen, was dazu führt, dass er auf einen Aufruf pro Sekunde gedrosselt wird, es sei denn, Sie beenden den Aufruf der API, um den Bucket wieder auffüllen zu lassen.

Hier sind die Tarife für Device Farm APIs:

- Für List and Get APIs beträgt die Kapazität der Burst-API-Anfragen`50`, und die Transactions-per-second (TPS-) Rate ist`10`.
- Für alle anderen APIs beträgt `10` die Kapazität der Burst-API-Anfragen und die Transactions-per-second (TPS-) Rate ist. 1

Appium-Endpunktgrenzen

Die folgenden Grenzwerte gelten für alle Appium-Endpunktsitzungen. Für Fragen und Anleitungen zum besten Umgang mit Limits wenden Sie sich bitte an einen Support-Fall.

- Jeder Appium-Befehl hat eine maximale Ausführungsdauer von 4 Minuten. Danach wird das Timeout für den Befehl überschritten.
- Der Endpunkt akzeptiert Eingabe-Payload-Größen von bis zu 20 MB und erlaubt Ausgabe-Payload-Größen von bis zu 20 MB. Bei jeder Anfrage mit einer größeren Eingabe- oder Ausgabegröße wird eine Fehlermeldung von angezeigt. WebDriver `'unsupported operation'`
- Anfragen werden sequentiell auf dem Gerät in der Reihenfolge ausgeführt, in der sie empfangen wurden. Daher empfehlen wir dringend, Befehle nacheinander zu senden und auf die Antwort jedes Befehls zu warten, bevor ein neuer gesendet wird. Allerdings können bestimmte Appium-Serverbefehle parallel gesendet werden, insbesondere:
  - Status abrufen
  - Sitzungen abrufen
- Der Endpunkt unterstützt das WebDriver BiDi Protokoll derzeit nicht.

- Der Endpunkt unterstützt keine Appium-Plugins oder Treiber außer den XCUITest UIAutomator2 UND-Treibern.
- Maximal 3 Apps können als Hilfs-Apps mit einer Anfrage zur Erstellung einer Fernzugriffssitzung verwendet werden. Allerdings gibt es keine Begrenzung, wie viele Apps während einer Sitzung mithilfe der `InstallToRemoteAccessSession`API installiert werden können.

Grenzwerte für benutzerdefinierte Umgebungsvariablen

Die folgenden Grenzwerte gelten für alle benutzerdefinierten Umgebungsvariablen. Wenn Sie Fragen und Anleitungen zum optimalen Umgang mit Grenzwerten haben, wenden Sie sich bitte an einen Support-Fall.

- Für ein bestimmtes Device Farm Farm-Projekt oder eine Ausführung können maximal 32 Variablen konfiguriert werden.
- Variablennamen dürfen nicht länger als 256 Zeichen sein.
- Variablennamen unterliegen den Einschränkungen von`bash`. Sie dürfen nämlich nur alphanumerische Zeichen und Unterstriche enthalten und dürfen nicht mit einer Zahl beginnen.
- Variablennamen, die mit beginnen, `$DEVICEFARM_` sind für die interne Verwendung durch Dienste reserviert.
- Variablenwerte dürfen eine Länge von 256 Zeichen nicht überschreiten.
- Umgebungsvariablen können nicht verwendet werden, um die Computerauswahl für den Testhost in der Testspezifikationsdatei zu konfigurieren.

Tools und Plugins für AWS Device Farm

Dieser Abschnitt enthält Links und Informationen zur Arbeit mit den Tools und Plug-ins von AWS Device Farm. Device Farm Farm-Plug-ins finden Sie in den AWS-Laboren unter GitHub.

Wenn Sie ein Android-Entwickler sind, haben wir auch eine AWS Device Farm Farm-Beispiel-App für Android im Angebot GitHub. Sie können die App und die Beispieltests als Referenz für Ihre eigenen Device Farm Farm-Testskripte verwenden.

Themen

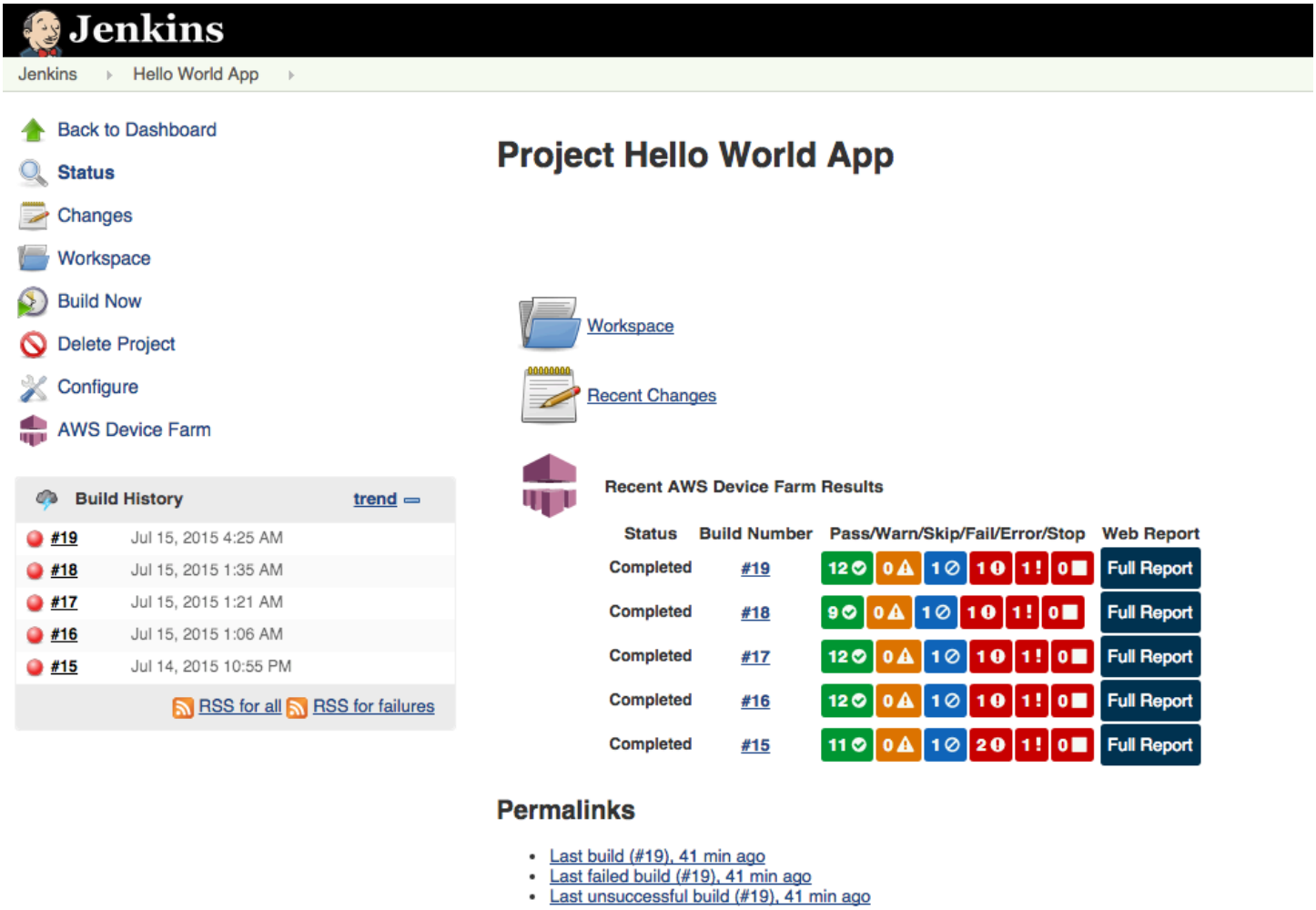
Integration von Device Farm mit einem Jenkins CI-Server

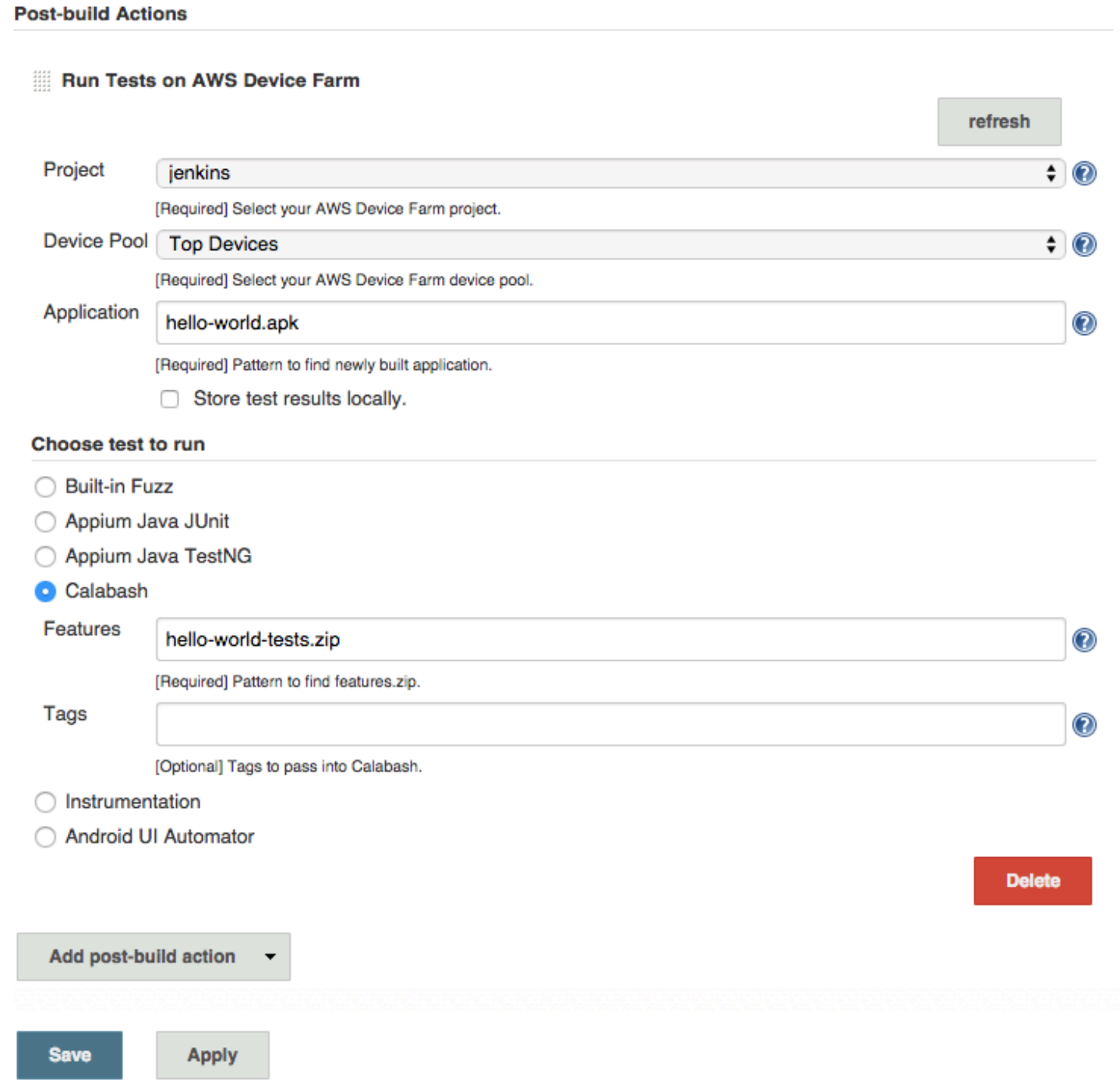
Das Jenkins CI-Plugin bietet AWS Device Farm Farm-Funktionalität von Ihrem eigenen Jenkins Continuous Integration (CI) -Server aus. Weitere Informationen finden Sie unter Jenkins (Software).

> **Note**
>
> Um das Jenkins-Plugin herunterzuladen, gehen Sie zu GitHubund folgen Sie den Anweisungen unter. Schritt 1: Installation des Jenkins CI-Plug-ins für AWS Device Farm

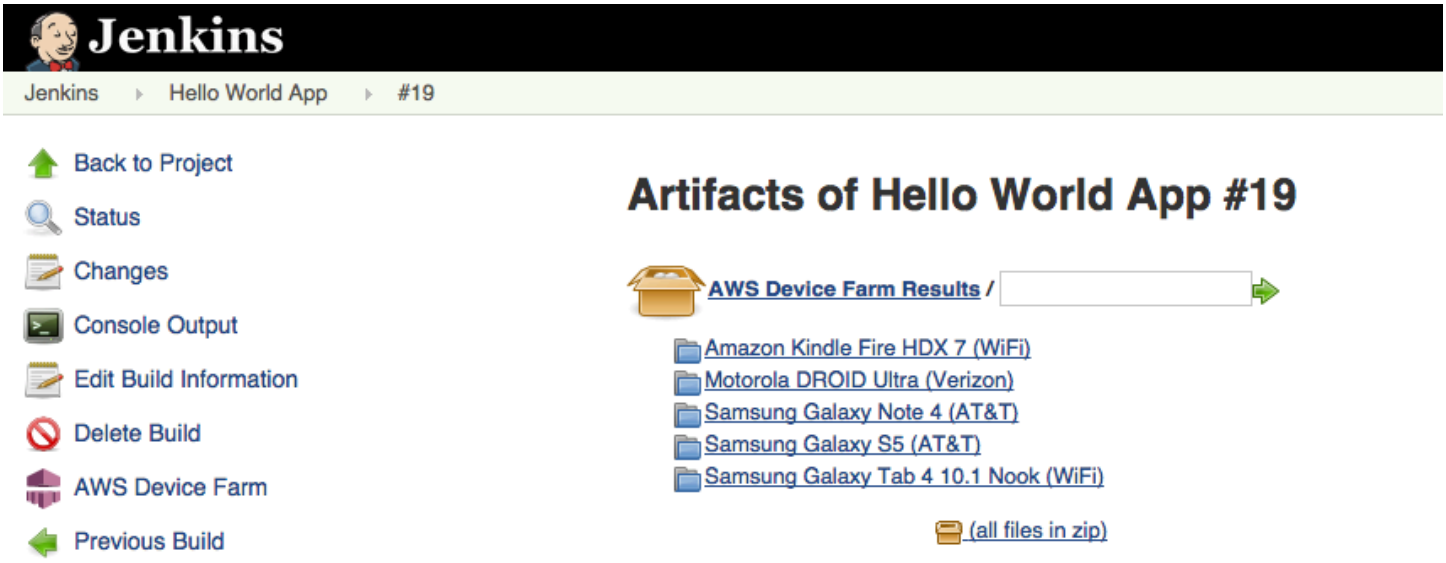
Dieser Abschnitt enthält eine Reihe von Verfahren zur Einrichtung und Verwendung des Jenkins CI-Plug-ins mit AWS Device Farm.

Die folgenden Bilder zeigen die Funktionen des Jenkins CI-Plug-ins.


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



Das Plug-in kann auch alle Testartefakte (Protokolle, Screenshots usw.) lokal abrufen:



Themen

- Abhängigkeiten
- Schritt 1: Installation des Jenkins CI-Plug-ins für AWS Device Farm
- Schritt 2: Einen AWS Identity and Access Management Benutzer für Ihr Jenkins CI Plugin für AWS Device Farm erstellen
- Schritt 3: Erstmaliges Konfigurieren des Jenkins CI-Plug-ins in AWS Device Farm
- Schritt 4: Verwenden des Plugins in einem Jenkins-Job

Abhängigkeiten

Das Jenkins CI-Plugin erfordert das AWS Mobile SDK 1.10.5 oder höher. Weitere Informationen und eine Anleitung zur Installation des SDK finden Sie unter AWS Mobile-SDK.

Schritt 1: Installation des Jenkins CI-Plug-ins für AWS Device Farm

Es gibt zwei Optionen für die Installation des Jenkins-Plug-ins Continuous Integration (CI) für AWS Device Farm. Sie können im Dialogfeld Available Plugins (Verfügbare Plug-ins) in der Web-Benutzeroberfläche von Jenkins nach dem Plug-in suchen oder die Datei `hpi` herunterladen und direkt aus Jenkins heraus installieren.

Installieren über die Jenkins-Benutzeroberfläche

1. Suchen Sie das Plug-ins über die Jenkins-Benutzeroberfläche, indem Sie nacheinander die Optionen Manage Jenkins (Jenkins verwalten), Manage Plugins (Plug-ins verwalten) und Available (Verfügbar) wählen.
2. Suchen Sie nach aws-device-farm.
3. Installieren Sie das AWS Device Farm Farm-Plug-In.
4. Stellen Sie sicher, dass der Besitzer der Datei der Benutzer `Jenkins` ist.
5. Starten Sie Jenkins neu.

Laden Sie das Plugin herunter

1. Laden Sie die `hpi` Datei direkt von http://updates.jenkins-ci herunter. org/latest/aws-device-farm.hpi.
2. Stellen Sie sicher, dass der Besitzer der Datei der Benutzer `Jenkins` ist.
3. Installieren Sie das Plug-in mithilfe einer der folgenden Verfahren:
   - Laden Sie das Plug-in hoch, indem Sie nacheinander die Optionen Manage Jenkins (Jenkins verwalten), Manage Plugins (Plug-ins verwalten), Advanced (Erweitert) und dann Upload plugin (Plug-in hochladen) wählen.
   - Verschieben Sie die Datei `hpi` in das Jenkins-Plug-in-Verzeichnis (in der Regel `/var/lib/jenkins/plugins`).
4. Starten Sie Jenkins neu.

Schritt 2: Einen AWS Identity and Access Management Benutzer für Ihr Jenkins CI Plugin für AWS Device Farm erstellen

Wir empfehlen, dass Sie Ihr AWS Root-Konto nicht für den Zugriff auf Device Farm verwenden. Erstellen Sie stattdessen einen neuen AWS Identity and Access Management (IAM-) Benutzer (oder verwenden Sie einen vorhandenen IAM-Benutzer) in Ihrem AWS Konto und greifen Sie dann mit diesem IAM-Benutzer auf Device Farm zu.

Informationen zum Erstellen eines neuen IAM-Benutzers finden Sie unter Einen IAM-Benutzer erstellen ().AWS-Managementkonsole Achten Sie darauf, dass Sie für jeden Benutzer einen Zugriffsschlüssel erstellen, und laden Sie die Sicherheitsanmeldeinformationen für jeden Benutzer

herunter oder speichern Sie diese. Sie benötigen die Anmeldeinformationen zu einem späteren Zeitpunkt.

Erteilen Sie dem IAM-Benutzer die Erlaubnis, auf Device Farm zuzugreifen

Um dem IAM-Benutzer Zugriff auf Device Farm zu gewähren, erstellen Sie eine neue Zugriffsrichtlinie in IAM und weisen Sie die Zugriffsrichtlinie dann dem IAM-Benutzer wie folgt zu.

> **Note**
>
> Das AWS Root-Konto oder der IAM-Benutzer, mit dem Sie die folgenden Schritte ausführen, muss berechtigt sein, die folgende IAM-Richtlinie zu erstellen und sie an den IAM-Benutzer anzuhängen. Weitere Informationen finden Sie unter Arbeiten mit Richtlinien

Um die Zugriffsrichtlinie in IAM zu erstellen

1. Öffnen Sie unter https://console.aws.amazon.com/iam/ die IAM-Konsole.
2. Wählen Sie Policies (Richtlinien).
3. Wählen Sie Richtlinie erstellen aus. (Wenn die Schaltfläche Get Started (Erste Schritte) angezeigt wird, klicken Sie darauf und wählen Sie anschließend Create Policy (Richtlinie erstellen) aus.)
4. Klicken Sie neben Create Your Own Policy auf Select.
5. Geben Sie unter Policy Name (Richtlinienname) einen Namen für die Richtlinie ein (z. B. **`AWSDeviceFarmAccessPolicy`**).
6. Geben Sie unter Beschreibung eine Beschreibung ein, anhand derer Sie diesen IAM-Benutzer Ihrem Jenkins-Projekt zuordnen können.
7. Geben Sie unter Policy Document (Richtliniendokument) die folgende Anweisung ein:

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "DeviceFarmAll",
            "Effect": "Allow",
```

```
            "Action": [ "devicefarm:*" ],
            "Resource": [ "*" ]
        }
    ]
}
```

8. Wählen Sie Richtlinie erstellen aus.

Um die Zugriffsrichtlinie dem IAM-Benutzer zuzuweisen

1. Öffnen Sie unter https://console.aws.amazon.com/iam/ die IAM-Konsole.
2. Wählen Sie Users (Benutzer) aus.
3. Wählen Sie den IAM-Benutzer aus, dem Sie die Zugriffsrichtlinie zuweisen möchten.
4. Wählen Sie im Bereich Permissions (Berechtigungen) unter Managed Policies (Verwaltete Richtlinien) die Option Attach Policy (Richtlinie anfügen).
5. Wählen Sie die gerade erstellte Richtlinie aus (z. B. AWSDeviceFarmAccessPolicy).
6. Wählen Sie Richtlinie anfügen aus.

Schritt 3: Erstmaliges Konfigurieren des Jenkins CI-Plug-ins in AWS Device Farm

Wenn Sie Ihrer Jenkins-Server zum ersten Mal ausführen möchten, müssen Sie das System wie folgt konfigurieren.

> **Note**
>
> Wenn Sie Geräteplätze verwenden, beachten Sie, dass das Geräteplatz-Feature standardmäßig deaktiviert ist.

1. Melden Sie sich an Ihrer Jenkins-Webbenutzeroberfläche an.
2. Wählen Sie auf der linken Seite des Bildschirms die Option Manage Jenkins (Jenkins verwalten) aus.
3. Wählen Sie Configure System (System konfigurieren) aus.
4. Scrollen Sie nach unten zum AWS Device Farm Farm-Header.

5. Kopieren Sie Ihre Anmeldeinformationen unter Einen IAM-Benutzer für Ihr Jenkins CI-Plugin erstellen und fügen Sie Ihre Zugriffsschlüssel-ID und den geheimen Zugriffsschlüssel in die jeweiligen Felder ein.
6. Wählen Sie Speichern.

Schritt 4: Verwenden des Plugins in einem Jenkins-Job

Verfahren Sie nach der Installation der Jenkins-Plug-ins wie folgt, um es in einem Jenkins-Auftrag zu verwenden.

1. Melden Sie sich bei Ihrer Jenkins-Webbenutzeroberfläche an.
2. Klicken Sie auf den Auftrag, den Sie bearbeiten möchten.
3. Wählen Sie auf der linken Seite des Bildschirms die Option Configure (Konfigurieren) aus.
4. Führen Sie einen Bildlauf nach unten zur Überschrift Post-build Actions (Postbuild-Aktionen) durch.
5. Klicken Sie auf Post-Build-Aktion hinzufügen und wählen Sie Tests auf AWS Device Farm ausführen aus.
6. Willen Sie das Projekt aus, das verwendet werden soll.
7. Wählen Sie den Gerätepool aus, der verwendet werden soll.
8. Legen Sie fest, ob die Testartefakte (z. B. Protokolle und Screenshots) lokal archiviert werden sollen.
9. Geben Sie im Feld Application (Anwendung) den Pfad zu Ihrer kompilierten Anwendung an.
10. Wählen Sie den Test aus, den Sie ausführen möchten, und füllen Sie alle Pflichtfelder aus.
11. Wählen Sie Speichern.

Integration von Device Farm in ein Gradle-Build-System

Das Device Farm Gradle-Plugin ermöglicht die Integration von AWS Device Farm mit dem Gradle-Build-System in Android Studio. Weitere Informationen finden Sie unter Gradle.

> **Note**
>
> Um das Gradle-Plugin herunterzuladen, gehen Sie zu GitHubund folgen Sie den Anweisungen unter. Das Device Farm Gradle-Plugin erstellen

Das Device Farm Gradle Plugin bietet Device Farm Farm-Funktionen aus Ihrer Android Studio-Umgebung. Sie können Tests auf echten Android-Handys und -Tablets starten, die von Device Farm gehostet werden.

Dieser Abschnitt enthält eine Reihe von Verfahren zum Einrichten und Verwenden des Device Farm Gradle Plug-ins.

Themen

Abhängigkeiten

Laufzeit

- Das Device Farm Gradle Plugin benötigt das AWS Mobile SDK 1.10.15 oder höher. Weitere Informationen und eine Anleitung zur Installation des SDK finden Sie unter AWS Mobile-SDK.
- Android Tools Builder Test-API 0.5.2
- Apache Commons Lang3 3.3.4

For Unit Tests (Für Einheitentests)

- Testng 6.8.8
- Jmockit 1.19
- Android Gradle Tools 1.3.0

Schritt 1: Erstellen des AWS Device Farm Gradle-Plug-ins

Dieses Plugin ermöglicht die Integration von AWS Device Farm mit dem Gradle-Build-System in Android Studio. Weitere Informationen finden Sie unter Gradle.

> **Note**
>
> Das Erstellen des Plug-Ins ist optional. Das Plug-In wird über Maven Central veröffentlicht. Wenn Sie Gradle das direkte Herunterladen des Plug-Ins zulassen möchten, überspringen Sie diesen Schritt und springen Sie zu Schritt 2: Einrichten des AWS Device Farm Gradle-Plug-ins.

So führen Sie das Plug-In aus

1. Gehe zum Repository GitHubund klone es.
2. Erstellen Sie das Plug-In mithilfe von `gradle install`.

   Das Plug-in wird auf Ihrem lokalen Maven Repository installiert.

Nächster Schritt: Schritt 2: Einrichten des AWS Device Farm Gradle-Plug-ins

Schritt 2: Einrichten des AWS Device Farm Gradle-Plug-ins

Wenn noch nicht erfolgt, klonen Sie das Repository und installieren Sie das Plug-In mit dem hier beschriebenen Verfahren: Das Device Farm Gradle-Plugin erstellen.

So konfigurieren Sie das AWS Device Farm Gradle Plugin

1. Fügen Sie das Plug-In-Artefakt Ihrer Abhängigkeitsliste in `build.gradle` hinzu.

```
buildscript {

    repositories {
        mavenLocal()
        mavenCentral()
    }

    dependencies {
        classpath 'com.android.tools.build:gradle:1.3.0'
        classpath 'com.amazonaws:aws-devicefarm-gradle-plugin:1.0'
    }
}
```

2. Konfigurieren Sie das Plug-In in Ihrer `build.gradle`-Datei. Die folgende testspezifische Konfiguration dient als Anleitung:

```
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. Führen Sie Ihren Device Farm Farm-Test mit der folgenden Aufgabe aus:`gradle devicefarmUpload`.

   Die Build-Ausgabe druckt einen Link zur Device Farm Farm-Konsole aus, über die Sie Ihre Testausführung überwachen können.

Nächster Schritt: Generieren eines IAM-Benutzers im Device Farm Gradle-Plugin

Schritt 3: Generieren eines IAM-Benutzers im Device Farm Gradle-Plugin

AWS Identity and Access Management (IAM) hilft Ihnen bei der Verwaltung von Berechtigungen und Richtlinien für die Arbeit mit Ressourcen. AWS Dieses Thema führt Sie durch die Generierung eines IAM-Benutzers mit Berechtigungen für den Zugriff auf AWS Device Farm Farm-Ressourcen.

Falls Sie dies noch nicht getan haben, führen Sie die Schritte 1 und 2 aus, bevor Sie einen IAM-Benutzer generieren.

Wir empfehlen, dass Sie Ihr AWS Root-Konto nicht für den Zugriff auf Device Farm verwenden. Erstellen Sie stattdessen einen neuen IAM-Benutzer (oder verwenden Sie einen vorhandenen IAM-Benutzer) in Ihrem AWS Konto und greifen Sie dann mit diesem IAM-Benutzer auf Device Farm zu.

> **Note**
>
> Das AWS Root-Konto oder der IAM-Benutzer, mit dem Sie die folgenden Schritte ausführen, muss berechtigt sein, die folgende IAM-Richtlinie zu erstellen und sie an den IAM-Benutzer anzuhängen. Weitere Informationen finden Sie unter Arbeiten mit Richtlinien.

Um einen neuen Benutzer mit der richtigen Zugriffsrichtlinie in IAM zu erstellen

1. Öffnen Sie unter https://console.aws.amazon.com/iam/ die IAM-Konsole.
2. Wählen Sie Users (Benutzer) aus.
3. Wählen Sie Create New Users (Neue Benutzer erstellen) aus.
4. Geben Sie den Benutzernamen Ihrer Wahl ein.

   Beispiel, **`GradleUser`**.
5. Wählen Sie Erstellen aus.
6. Wählen Sie Download Credentials (Download-Anmeldeinformationen) aus und speichern Sie die Anmeldeinformationen an einem Ort, von dem Sie sie später leicht abrufen können.
7. Wählen Sie Close (Schließen) aus.
8. Wählen Sie den Benutzernamen in der Liste aus.
9. Erweitern Sie unter Permissions (Berechtigungen) die Überschrift Inline Policies (Eingebundene Richtlinien), indem Sie auf den Pfeil nach unten auf der rechten Seite klicken.
10. Wählen Sie Klicken Sie hier, wo es heißt: Es sind keine Inline-Richtlinien zum Anzeigen vorhanden. Um eine zu erstellen, klicken Sie hier.

11. Wählen Sie auf der Seite Set Permissions (Berechtigungen festlegen) die Option Custom Policy (Benutzerdefinierte Richtlinie) aus.
12. Wählen Sie Select (Auswählen).
13. Geben Sie einen Namen für die Richtlinie ein, z. B. **`AWSDeviceFarmGradlePolicy`**.
14. Fügen Sie die folgende Richtlinie unter Policy Document (Richtliniendokument) ein.

    JSON

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

15. Klicken Sie auf Apply Policy (Richtlinie anwenden).

Nächster Schritt: Konfiguration von Testtypen.

Weitere Informationen finden Sie unter Einen IAM-Benutzer erstellen (AWS-Managementkonsole) oderEinrichtung.

Schritt 4: Testtypen konfigurieren

Standardmäßig führt das AWS Device Farm Gradle-Plugin den Instrumentierung für Android und AWS Device Farm Test aus. Wenn Sie eigene Tests ausführen oder zusätzliche Parameter angeben möchten, können Sie einen Testtyp konfigurieren. In diesem Thema finden Sie Informationen zu allen verfügbaren Testtypen und wie Sie in Android Studie vorgehen müssen, um sie für die Verwendung zu konfigurieren. Weitere Informationen zu den verfügbaren Testtypen in Device Farm finden Sie unterTest-Frameworks und integrierte Tests in AWS Device Farm.

Wenn noch nicht erfolgt, führen die Schritte 1 bis 3 aus, bevor Sie Testtypen konfigurieren.

> **Note**
>
> Wenn Sie Geräteplätze verwenden, beachten Sie, dass das Geräteplatz-Feature standardmäßig deaktiviert ist.

Appium

Device Farm bietet Unterstützung für Appium Java JUnit und TestNG für Android.

- Appium (unter Java ()) JUnit
- Appium (unter Java (TestNG))

Sie können `useTestNG()` oder `useJUnit()` auswählen. `JUnit` ist der Standardwert und muss nicht explizit angegeben werden.

```
appium {
    tests file("path to zip file") // required
    useTestNG() // or useJUnit()
}
```

Eingebaut: Fuzz

Device Farm bietet einen integrierten Fuzz-Testtyp, der nach dem Zufallsprinzip Benutzeroberflächenereignisse an Geräte sendet und dann die Ergebnisse meldet.

```
fuzz {

   eventThrottle 50 // optional default
   eventCount 6000  // optional default
   randomizerSeed 1234 // optional default blank

 }
```

Weitere Informationen finden Sie unter Ausführen des integrierten Fuzz-Tests von Device Farm (Android und iOS).

Instrumentierung

Device Farm bietet Unterstützung für Instrumentierung (EspressoJUnit, Robotium oder andere instrumentationsbasierte Tests) für Android. Weitere Informationen finden Sie unter Instrumentierung für Android und AWS Device Farm.

Wenn Sie einen Instrumentierungstest in Gradle ausführen, verwendet Device Farm die aus Ihrem AndroidTest-Verzeichnis generierte `.apk` Datei als Quelle für Ihre Tests.

```
    instrumentation {

        filter "test filter per developer docs" // optional

    }
```

AWS Device Farm Farm-Dokumentenverlauf

Die folgende Tabelle enthält wichtige Änderungen an der Dokumentation seit der letzten Veröffentlichung dieses Handbuchs.

Änderungen	Beschreibung	Änderungsdatum
Unterstützung für Appium-Endgeräte	Device Farm bietet jetzt einen vollständig verwalteten Appium-Endpunkt für Remote-Gerätetests, der eine schnelle Testentwicklung und ein schnelles Debuggen ermöglicht. Dies ergänzt die bestehende serverseitige Ausführungsmethode, bei der Tests hochgeladen und direkt auf Device Farm ausgeführt werden. Die serversei tige Ausführung ist zwar ideal für CI/CD Pipelines und groß angelegte Tests, der neue lokale Appium-Endpunkt ermöglicht jedoch eine schnellere Iteration und Entwicklung von Tests auf realen Geräten.	17. November 2025
Verbesserungen am iOS-Testhost	Device Farm unterstützt jetzt ein aktualisiertes Erlebnis für die iOS-Testumgebung und ermöglicht so Konsistenz in den Setups zwischen Android- und iOS-Tests. Weitere Informationen hierzu finden Sie unter Hosts für benutzerd efinierte Testumgebungen. Darüber hinaus wurden Informationen zu heruntergezogenen Android-Testhosts entfernt. Android-Benutzern wird empfohlen, die Amazon Linux 2-Testhosts zu verwenden.	31. Oktober 2025
AL2 Unterstützung	Device Farm unterstützt jetzt die AL2 Testumgebung für Android. Weitere Informationen zu AL2.	6. November 2023
Migration von Standard- zu benutzerdefinierten Testumgebungen	Im Dezember 2023 wurde der Migrationsleitfaden aktualisiert, der zeigt, dass Tests im Standardmodus nicht mehr unterstützt werden.	3. September 2023

Änderungen	Beschreibung	Änderungsdatum
VPC ENI-Unterstützung	Device Farm ermöglicht es nun privaten Geräten, die VPC-ENI-Konnektivitätsfunktion zu nutzen, um Kunden dabei zu unterstützen, eine sichere Verbindung zu ihren privaten Endpunkten herzustellen, die auf AWS, On-Premise-Software oder einem anderen Cloud-Anbieter gehostet werden. Erfahren Sie mehr über VPC-ENI.	15. Mai 2023
Aktualisierungen der Polaris-Benutzeroberfläche	Die Device Farm Farm-Konsole unterstützt jetzt das Polaris-Framework.	28. Juli 2021
Python 3-Unterstützung	Device Farm unterstützt jetzt Python 3 in Tests im benutzerdefinierten Modus. Hier erfahren Sie mehr über die Verwendung von Python 3 in Ihren Testpaketen: • Appium (Python) • Appium (Python)	20. April 2020
Neue Sicherheitsinformationen und Informationen zum Markieren von AWS Ressourcen.	Um die Sicherung von AWS Diensten einfacher und umfassender zu gestalten, wurde ein neuer Abschnitt zum Thema Sicherheit erstellt. Weitere Informationen finden Sie unter Sicherheit in AWS Device Farm. Ein neuer Abschnitt zum Tagging in Device Farm wurde hinzugefügt. Weitere Informationen zum Taggen finden Sie unter Taggen in der Device Farm.	27. März 2020
Entfernung des direkten Gerätezugriffs.	Direkter Gerätezugriff (Remote-Debugging auf privaten Geräten) ist nicht mehr für die allgemeine Nutzung verfügbar. Wenn Sie Fragen zur künftigen Verfügbarkeit des direkten Gerätezugriffs haben, wenden Sie sich an uns.	9. September 2019

Änderungen	Beschreibung	Änderungsdatum
Aktualisieren der Gradle-Plug-in-Konfiguration	Eine überarbeitete Gradle-Plugin-Konfiguration enthält jetzt eine anpassbare Version der Gradle-Konfiguration mit optionalen Parametern, die auskommentiert sind. Weitere Informationen zu Einrichtung des Device Farm Gradle-Plugins.	16. August 2019
Neue Anforderung für Testläufe mit XCTest	Für Testläufe, die das XCTest Framework verwenden, benötigt Device Farm jetzt ein App-Paket, das für Tests erstellt wurde. Weitere Informationen zu the section called “XCTest”.	4. Februar 2019
Unterstützung für Appium Node.js und Appium Ruby-Testtypen in benutzerdefinierten Umgebungen	Sie können Ihre Tests jetzt in benutzerdefinierten Appium Node.js- und Appium Ruby-Testumgebungen ausführen. Weitere Informationen zu Test-Frameworks und integrierte Tests in AWS Device Farm.	10. Januar 2019
Unterstützung für Appium-Server Version 1.7.2 in sowohl Standard- als auch benutzerdefinierten Umgebungen. Unterstützung für Version 1.8.1 mithilfe einer benutzerdefinierten Test-Spezifikation-YAML-Datei in einer benutzerdefinierten Testumgebung.	Mit den Appium Server-Versionen 1.72, 1.71 und 1.6.5 können Sie Ihre Tests jetzt sowohl in Standard- als auch benutzerdefinierten Testumgebungen ausführen. Sie können Ihre Tests auch mit den Versionen 1.8.1 und 1.8.0 unter Verwendung einer benutzerdefinierten Test-Spezifikation-YAML-Datei in einer benutzerdefinierten Testumgebung ausführen. Weitere Informationen zu Test-Frameworks und integrierte Tests in AWS Device Farm.	2. Oktober 2018

Änderungen	Beschreibung	Änderungsdatum
Benutzerdefinierte Testumgebungen	Mit einer benutzerdefinierten Testumgebung können Sie sicherstellen, dass Ihre Tests wie in Ihrer lokalen Umgebung ausgeführt werden. Device Farm bietet jetzt Unterstützung für Live-Protokoll und Videostreaming, sodass Sie sofortiges Feedback zu Ihren Tests erhalten, die in einer benutzerdefinierten Testumgebung ausgeführt werden. Weitere Informationen zu Benutzerdefinierte Testumgebungen in AWS Device Farm.	16. August 2018
Support für die Verwendung von Device Farm als AWS CodePipeline Testanbieter	Sie können jetzt eine Pipeline so konfigurieren AWS CodePipeline , dass sie AWS Device Farm Farm-Läufe als Testaktionen in Ihrem Release-Prozess verwendet. CodePipeline ermöglicht es Ihnen, Ihr Repository schnell mit Build- und Testphasen zu verknüpfen, um ein kontinuierliches Integrationssystem zu erreichen, das auf Ihre Bedürfnisse zugeschnitten ist. Weitere Informationen zu Integration von AWS Device Farm in einer CodePipeline Testphase.	19. Juli 2018
Unterstützung privater Geräte	Sie können jetzt private Geräte verwenden, um Testläufe einzuplanen und Sitzungen mit Fernzugriff zu starten. Sie können Profile und Einstellungen für diese Geräte verwalten, Amazon VPC-Endpunkte zum Testen privater Apps erstellen und Remote-Debugging-Sitzungen erstellen . Weitere Informationen zu Private Geräte in AWS Device Farm.	2. Mai 2018
Unterstützung für Appium 1.6.3	Sie können jetzt die Appium-Version für Ihre benutzerdefinierten Appium-Tests festlegen.	21. März 2017
Festlegen des Ausführungstimeouts für einen Testlauf	Sie können die Zeitbeschränkung für die Ausführung eines Testlaufs oder aller Testläufe in einem Projekt festlegen . Weitere Informationen zu Einstellung des Ausführungszeitlimits für Testläufe in AWS Device Farm.	9. Februar 2017

Änderungen	Beschreibung	Änderungsdatum
Traffic Shaping	Sie können jetzt Netzwerkverbindungen und -bedingungen für einen Testlauf simulieren. Weitere Informationen zu Simulation von Netzwerkverbindungen und -bedingungen für Ihre AWS Device Farm Farm-Läufe.	8. Dezember 2016
Neuer Abschnitt für Fehlerbehebung	Sie können jetzt Probleme beim Hochladen von Testpaketen mithilfe einer Reihe von Verfahren beheben, die darauf ausgelegt sind, Fehlermeldungen zu beheben, die möglicherweise in der Device Farm Farm-Konsole auftreten. Weitere Informationen zu Behebung von Gerätefarm-Fehlern.	10. August 2016
Remotezugriffssitzungen	Sie können remote auf einzelne Gerät in der Konsole zugreifen und mit den Geräten interagieren. Weitere Informationen zu Fernzugriff.	19. April 2016
Geräteplätze-Self-Service	Sie können jetzt Geräteslots mit der AWS-Managementkonsole AWS Command Line Interface, oder der API erwerben. Weitere Informationen finden Sie unter Kauf eines Geräteslots in Device Farm.	22. März 2016
So stoppen Sie Testläufe	Sie können jetzt Testläufe mithilfe der AWS-Managementkonsole AWS Command Line Interface, der oder der API beenden. Weitere Informationen finden Sie unter Stoppen eines Laufs in AWS Device Farm.	22. März 2016
Neue XCTest UI-Testtypen	Sie können jetzt benutzerdefinierte XCTest UI-Tests für iOS-Anwendungen ausführen. Weitere Informationen über den Testtyp finden Sie unter Integrieren der XCTest Benutzeroberfläche für iOS mit Device Farm.	8. März 2016
Neue Appium Python-Testtypen	Sie können jetzt benutzerdefinierte Appium Python-Tests über Android-, iOS- und Webanwendungen ausführen. Weitere Informationen zu Test-Frameworks und integrierte Tests in AWS Device Farm.	19. Januar 2016

Änderungen	Beschreibung	Änderungsdatum
Testtypen für Webanwendungen	Sie können jetzt benutzerdefinierte Appium Java JUnit - und TestNG-Tests für Webanwendungen ausführen. Weitere Informationen zu Web-App-Tests in AWS Device Farm.	19. November 2015
Gradle-Plug-In für AWS Device Farm	Hier erfahren Sie mehr über die Installation und den Betrieb von Device Farm Gradle-Plugin.	28. September 2015
Neuer integrierter Test: Explorer	Bei dem Explorer-Test wird Ihre App durchlaufen, indem jeder einzelne Bildschirm aus der Sicht eines Endbenutzers analysiert und Screenshots erstellt werden.	16. September 2015
Unterstützung für iOS hinzugefügt	Weitere Informationen zum Testen von iOS-Geräten und zum Ausführen von iOS-Tests (einschließlich XCTest) finden Sie unterTest-Frameworks und integrierte Tests in AWS Device Farm.	4. August 2015
Erste veröffentlichte Version	Dies ist die erste öffentliche Version des AWS Device Farm Developer Guide.	13. Juli 2015

AWS Glossar

Die neueste AWS Terminologie finden Sie im AWS Glossar in der AWS-Glossar Referenz.

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.