Guida per gli sviluppatori

AWS Device Farm

Versione API 2015-06-23

AWS Device Farm: Guida per gli sviluppatori

Table of Contents

Cos'è AWS Device Farm?

Device Farm è un servizio di test di app che puoi utilizzare per testare e interagire con le tue app Android, iOS e Web su telefoni e tablet fisici reali ospitati da Amazon Web Services (AWS).

Esistono due modi principali per utilizzare Device Farm:

- Accedi in remoto a un dispositivo dal tuo computer locale, in modo interattivo nel tuo browser web o testandolo automaticamente utilizzando Appium da un client locale.
- Esegui automaticamente i test delle app utilizzando l'ambiente di esecuzione dei test gestito di Device Farm.

> **Note**
>
> Device Farm è disponibile solo nella regione `us-west-2` (Oregon).

accesso remoto

L'accesso remoto consente di interagire con un dispositivo tramite il browser Web in tempo reale. L'accesso remoto consente inoltre di eseguire test Appium dal client locale su dispositivi Device Farm remoti utilizzando un endpoint Appium gestito.

L'interazione in tempo reale con un dispositivo può essere utile per diversi scenari, come il test manuale delle app, la riproduzione di bug su un dispositivo specifico, il controllo della resa visiva dell'app su diversi tipi di schermo e le sequenze di installazione e aggiornamento delle app. L'endpoint Appium completamente gestito di Device Farm consente di sviluppare, testare ed eseguire il debug dei test Appium, fornendo un feedback rapido.

L'endpoint Appium supporta qualsiasi lingua di tua scelta, qualsiasi IDE locale, debug live con breakpoint, video e registri live e strumenti come Appium Inspector. Puoi eseguire i test tutte le volte che vuoi sullo stesso dispositivo durante la sessione di accesso remoto con un limite di 150 minuti.

Durante una sessione di accesso remoto, Device Farm registra i dettagli sulle azioni che avvengono durante l'interazione con il dispositivo. Alla fine della sessione sono prodotti log con i dettagli e un'acquisizione video della sessione.

Test automatizzato delle app

Device Farm ti consente di eseguire test automatici su più dispositivi in parallelo caricando l'app e i test. I test vengono eseguiti automaticamente in un ambiente completamente gestito su host di test in cui è possibile configurare un file di specifiche di test. L'ambiente utilizza gli host di test di Device Farm, quindi non devi preoccuparti di predisporre la tua infrastruttura per l'esecuzione dei test. Gli host e i dispositivi di test possono connettersi in modo sicuro al tuo VPC per accedere ai tuoi endpoint privati.

Una volta completati i test, viene generato un rapporto di test che contiene risultati di alto livello, log di basso livello, schermate e artefatti del test.

Device Farm supporta il test di app Android e iOS native e ibride. Per ulteriori informazioni sui tipi di test supportati, consulta Framework di test e test integrati in AWS Device Farm.

Terminologia

Device Farm introduce i seguenti termini che definiscono il modo in cui le informazioni sono organizzate:

pool di dispositivi

> Una serie di dispositivi che in genere condividono caratteristiche simili come piattaforma, produttore o modello.

job

> Una richiesta a Device Farm di testare una singola app su un singolo dispositivo. Un processo contiene una o più suite.

misurazione

> Si riferisce alla fatturazione per i dispositivi. Potresti trovare riferimenti a dispositivi misurati o dispositivi non misurati nella documentazione e nei riferimenti delle API. Per ulteriori informazioni sui prezzi, consulta la pagina dei prezzi di AWS Device Farm.

project

> Un'area di lavoro logica che contiene sessioni, una sessione per ogni test di una singola app in uno o più dispositivi. Si possono utilizzare i progetti per organizzare le aree di lavoro secondo le proprie preferenze. Ad esempio, è possibile avere un progetto per titolo di app oppure un progetto per piattaforma. È possibile creare tutti i progetti necessari.

report

Contiene informazioni su un'esecuzione, ovvero una richiesta a Device Farm di testare una singola app su uno o più dispositivi. Per ulteriori informazioni, consulta Report in AWS Device Farm.

run

Una specifica build dell'app, con un set specifico di test, da eseguire su un set specifico di dispositivi. Una sessione genera un report dei risultati. Un'esecuzione contiene uno o più processi. Per ulteriori informazioni, consulta Esecuzioni.

sessione

Un'interazione in tempo reale con un dispositivo fisico reale tramite browser Web. Per ulteriori informazioni, consulta Sessioni.

suite

L'organizzazione gerarchica dei test in un pacchetto di test. Una suite contiene uno o più test.

test

Un singolo test case in un pacchetto di test.

Per ulteriori informazioni su Device Farm, consulta Concetti.

Configurazione

Per utilizzare Device Farm, vedereConfigurazione.

Configurazione di AWS Device Farm

Prima di utilizzare Device Farm per la prima volta, è necessario completare le seguenti attività:

Argomenti

Passaggio 1: iscriviti a AWS

Iscriviti ad Amazon Web Services (AWS).

Se non ne possiedi uno Account AWS, completa i seguenti passaggi per crearne uno.

Per iscriverti a un Account AWS

1. Apri la https://portal.aws.amazon.com/billing/registrazione.
2. Segui le istruzioni online.

   Nel corso della procedura di registrazione riceverai una telefonata o un messaggio di testo e ti verrà chiesto di inserire un codice di verifica attraverso la tastiera del telefono.

   Quando ti iscrivi a un Account AWS, Utente root dell'account AWSviene creato un. L'utente root dispone dell'accesso a tutte le risorse e tutti i Servizi AWS nell'account. Come best practice di sicurezza, assegna l'accesso amministrativo a un utente e utilizza solo l'utente root per eseguire attività che richiedono l'accesso di un utente root.

Passaggio 2: crea o utilizza un utente IAM nel tuo AWS account

Si consiglia di non utilizzare l'account AWS root per accedere a Device Farm. Invece, crea un utente AWS Identity and Access Management (IAM) (o usane uno esistente) nel tuo AWS account, quindi accedi a Device Farm con quell'utente IAM.

Per ulteriori informazioni, consulta Creazione di un utente IAM (Console di gestione AWS).

Passaggio 3: concedere all'utente IAM l'autorizzazione ad accedere a Device Farm

Concedi all'utente IAM l'autorizzazione ad accedere a Device Farm. A tale scopo, crea una policy di accesso in IAM, quindi assegna la policy di accesso all'utente IAM, come segue.

> **Note**
>
> L'account AWS root o l'utente IAM che utilizzi per completare i seguenti passaggi deve disporre dell'autorizzazione per creare la seguente policy IAM e collegarla all'utente IAM. Per ulteriori informazioni, consulta l'articolo relativo all'utilizzo delle policy.

1. Crea una policy con il seguente codice JSON. Assegnagli un titolo descrittivo, ad esempio*DeviceFarmAdmin*.

   Per ulteriori informazioni sulla creazione di policy IAM, consulta Creating IAM Policies nella IAM User Guide.
2. Allega la policy IAM che hai creato al tuo nuovo utente. Per ulteriori informazioni su come allegare le policy IAM agli utenti, consulta Aggiungere e rimuovere le policy IAM nella IAM User Guide.

L'aggiunta della policy fornisce all'utente IAM l'accesso a tutte le azioni e le risorse di Device Farm associate a quell'utente IAM. Per informazioni su come limitare gli utenti IAM a un insieme limitato di azioni e risorse di Device Farm, consultaGestione delle identità e degli accessi in AWS Device Farm.

Approfondimenti

Ora sei pronto per iniziare a usare Device Farm. Per informazioni, consulta Guida introduttiva a Device Farm.

Guida introduttiva a Device Farm

Questa procedura dettagliata mostra come utilizzare Device Farm per testare un'app nativa per Android o iOS. La console Device Farm viene utilizzata per creare un progetto, caricare un file.apk o .ipa, eseguire una suite di test standard e quindi visualizzare i risultati.

> **Note**
>
> Device Farm è disponibile solo nella AWS regione `us-west-2` (Oregon).

Argomenti

Prerequisiti

Prima di iniziare, assicurati di aver completato i seguenti requisiti:

- Completa le fasi descritte in Configurazione. Sono necessari un AWS account e un utente AWS Identity and Access Management (IAM) con autorizzazione per accedere a Device Farm.

- Per Android, puoi importare un file.apk (pacchetto app Android) o utilizzare l'applicazione di esempio che forniamo. Per iOS, è necessario disporre di un file .ipa (archivio di app iOS). Caricherai il file su Device Farm più avanti in questa procedura dettagliata.

  > **Note**
  >
  > Assicurati che il file .ipa sia integrato per un dispositivo iOS e non per un simulatore.

- (Facoltativo) È necessario un test da uno dei framework di test supportati da Device Farm. Carichi questo pacchetto di test su Device Farm, quindi esegui il test più avanti in questa procedura dettagliata. Se non disponi di un pacchetto di test disponibile, puoi specificare ed eseguire una suite di test standard integrata. Per ulteriori informazioni, consulta Framework di test e test integrati in AWS Device Farm.

Fase 1: Accesso alla console di

Puoi utilizzare la console Device Farm per creare e gestire progetti ed esecuzioni per i test. In seguito in questa procedura, otterrai informazioni sui progetti e sulle sessioni.

- Accedere alla console Device Farm all'indirizzo https://console.aws.amazon.com/devicefarm.

Fase 2: Creare un progetto

Per testare un'app in Device Farm, devi prima creare un progetto.

1. Nel riquadro di navigazione, scegli Mobile Device Testing, quindi scegli Progetti.
2. In Progetti di test per dispositivi mobili, scegli Crea progetto.
3. In Crea progetto, inserisci un nome di progetto (ad esempio,**`MyDemoProject`**).
4. Scegli Create (Crea).

   La console apre la pagina Test automatici del progetto appena creato.

Fase 3: Creare e avviare una corsa

Ora che disponi di un progetto, puoi creare e avviare una sessione. Per ulteriori informazioni, consulta Esecuzioni.

1. Nella scheda Test automatici, scegli Crea esecuzione. In alternativa, puoi seguire il tutorial integrato nella console selezionando Crea esegui con tutorial.
2. (Facoltativo) In Impostazioni di esecuzione, nella sezione Nome esecuzione, inserisci un nome per la corsa. Se non viene fornito alcun nome, la console Device Farm chiamerà l'esecuzione «My Device Farm run» per impostazione predefinita.

3. In Impostazioni di esecuzione, nella sezione Tipo di esecuzione, seleziona il tipo di corsa. Seleziona l'app per Android se non hai un'app pronta per il test o se stai testando un'app per Android (.apk). Seleziona App iOS se stai testando un'app iOS (.ipa).
4. In Seleziona app, nella sezione Opzioni di selezione app, scegli Seleziona app di esempio fornita da Device Farm se non hai un'app disponibile per il test. Se porti la tua app, seleziona Carica la tua app e scegli il file dell'applicazione. Se si sta caricando un'app iOS, assicurarsi di selezionare iOS device (dispositivo iOS), invece di un simulatore.
5. In Configura test, nella sezione Seleziona framework di test, scegli uno dei framework di test o delle suite di test integrate. Per ulteriori informazioni su ciascuna opzione, consulta Framework di test e test integrati.
   - Se non hai ancora confezionato i test per Device Farm, scegli Built-in: Fuzz per eseguire una suite di test standard integrata. Puoi mantenere i valori predefiniti per Event count, Event throttle e Randomizer seed. Per ulteriori informazioni, consulta the section called “Integrato: fuzz (Android e iOS)”.
   - Se disponi di un pacchetto di test da uno dei framework di test supportati, scegli il framework di test corrispondente e carica il file che contiene i test.
6. In Seleziona dispositivi, scegli Usa Device Pool e Top Devices.
7. (Facoltativo) Per aggiungere una configurazione aggiuntiva, apri il menu a discesa Configurazione aggiuntiva. In questa sezione, puoi effettuare una delle seguenti operazioni:
   - Per fornire altri dati da utilizzare a Device Farm durante l'esecuzione, accanto a Aggiungi dati aggiuntivi, scegli Scegli file, quindi cerca e scegli il file.zip che contiene i dati.
   - Per installare un'app aggiuntiva da utilizzare in Device Farm durante l'esecuzione, accanto a Installa altre app, scegli Scegli file, quindi cerca e scegli il file.apk o .ipa che contiene l'app. Ripetere questa operazione per altre applicazioni che si desidera installare. Puoi modificare l'ordine di installazione trascinando le app dopo averle caricate.
   - Per specificare se una rete Wi-Fi, Bluetooth, GPS o NFC sia abilitata durante la sessione, accanto a Set radio states (Imposta stati radio), selezionare le caselle appropriate.
   - Per preconfigurare la latitudine e la longitudine del dispositivo per la sessione, accanto a Device location (Posizione dispositivo), immettere le coordinate.
   - Per preimpostare le impostazioni locali del dispositivo per l'esecuzione, in Impostazioni locali del dispositivo, scegli le impostazioni locali.
   - Seleziona Abilita la registrazione video per registrare video durante il test.

- Seleziona Abilita l'acquisizione dei dati sulle prestazioni dell'app per acquisire i dati sulle prestazioni dal dispositivo.

> **Note**
>
> L'impostazione dello stato radio e delle impostazioni locali del dispositivo sono opzioni disponibili solo per i test nativi Android al momento.

> **Note**
>
> Se disponi di dispositivi privati, viene visualizzata anche la configurazione specifica per i dispositivi privati.

8. Nella parte inferiore della pagina, scegli Crea corsa per pianificare la corsa.

Device Farm avvia l'esecuzione non appena i dispositivi sono disponibili, in genere entro pochi minuti. Per visualizzare lo stato dell'esecuzione, nella pagina Test automatici del progetto, scegli il nome dell'esecuzione. Nella pagina di esecuzione, in Dispositivi, ogni dispositivo inizia con l'icona

in sospeso nella tabella dei dispositivi, quindi passa all'icona

in esecuzione all'inizio del test. Al termine di ogni test, la console visualizza l'icona del risultato del test accanto al nome del dispositivo. Una volta completati tutti i test, l'icona in sospeso accanto all'esecuzione diventa l'icona del risultato del test.

Passaggio 4: Visualizza i risultati della corsa

Per visualizzare i risultati dei test dell'esecuzione, nella pagina Test automatici del progetto, scegli il nome della corsa. Una pagina di riepilogo mostra:

- Il numero totale di test, in base al risultato.
- Elenchi di test con avvisi o errori univoci.
- Un elenco di dispositivi con i risultati dei test per ciascuno di essi.
- Qualsiasi screenshot acquisito durante la sessione, raggruppato per dispositivo.

- Una sezione per scaricare il risultato dell'analisi.

Per ulteriori informazioni, consulta Visualizzazione dei report dei test in Device Farm.

Fasi successive

Per ulteriori informazioni su Device Farm, consulta Concetti.

Acquisto di uno slot per dispositivi in Device Farm

È possibile utilizzare la console Device Farm, AWS Command Line Interface (AWS CLI) o l'API Device Farm per acquistare uno slot per dispositivi.

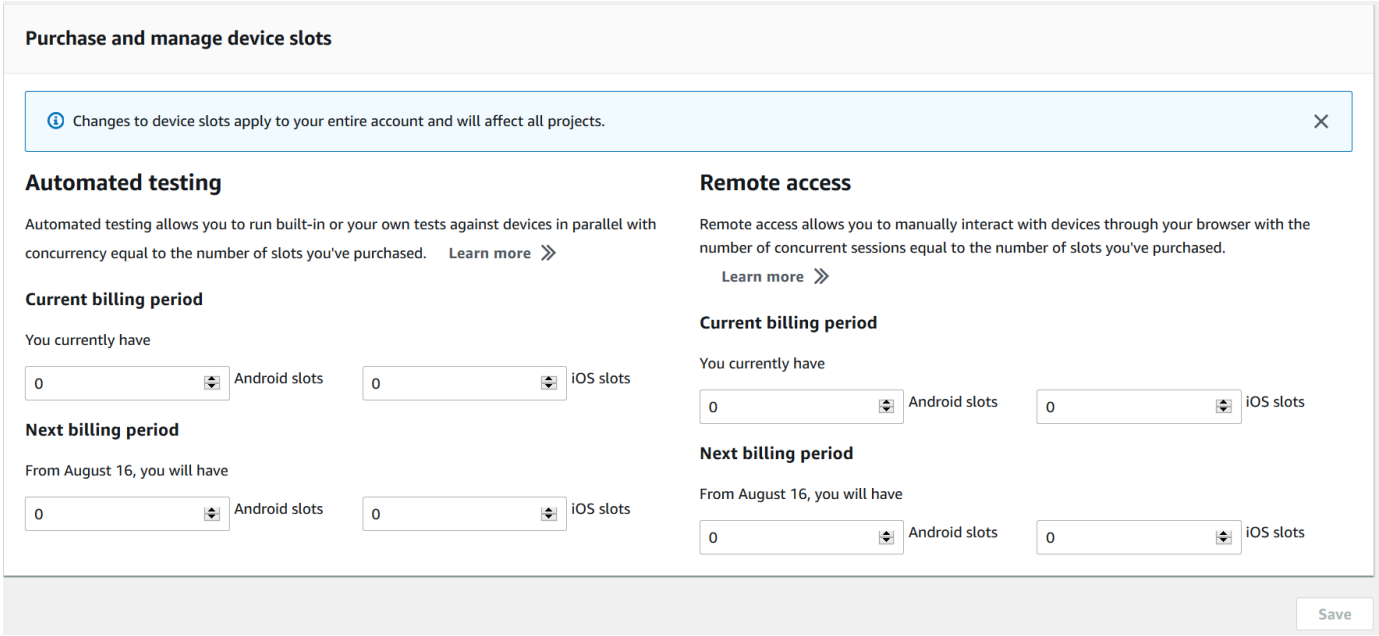
Acquista slot per dispositivi (console)

1. Accedere alla console Device Farm all'indirizzo https://console.aws.amazon.com/devicefarm.
2. Nel riquadro di navigazione, scegli Mobile Device Testing, quindi scegli Device slots.
3. Nella pagina Acquista e gestisci gli slot dei dispositivi, puoi creare il tuo pacchetto personalizzato scegliendo il numero di slot per dispositivi di test automatici e accesso remoto che desideri acquistare. Specificate gli importi degli slot per il periodo di fatturazione corrente e successivo.

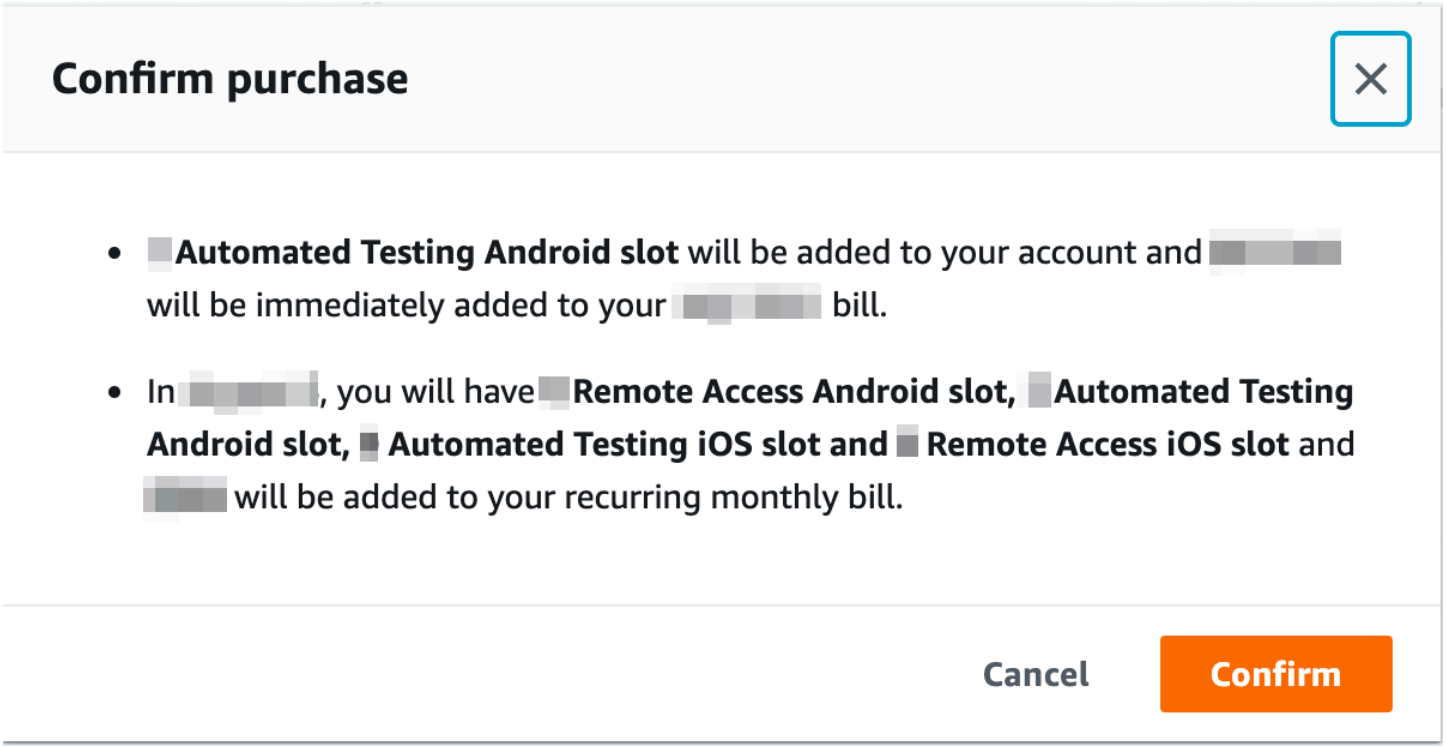
   Quando modifichi l'importo dello slot, il testo si aggiorna dinamicamente con l'importo di fatturazione. Per ulteriori informazioni, consulta i prezzi di AWS Device Farm.

   > **Important**
   >
   > Se modifichi il numero di slot per dispositivi ma visualizzi il messaggio «Contattaci» o «Contattaci per l'acquisto», significa che il tuo AWS account non è ancora autorizzato ad acquistare il numero di slot per dispositivi che hai richiesto.
   > Queste opzioni richiedono di inviare un'e-mail al team di supporto di Device Farm. Nell'e-mail, specifica il numero di ogni tipo di dispositivo che desideri acquistare e per quale ciclo di fatturazione.

   > **Note**
   >
   > Le modifiche agli slot dei dispositivi si applicano all'intero account e influiscono su tutti i progetti.



4. Scegliere Purchase (Acquista). Apparirà una finestra di conferma dell'acquisto. Controlla le informazioni, quindi scegli Conferma per completare la transazione.



Nella pagina Acquista e gestisci gli slot dei dispositivi, puoi vedere il numero di slot per dispositivi di cui disponi attualmente. Se si aumenta o diminuisce il numero di slot, si potrà visualizzare il numero di slot di cui si dispone un mese dopo la data della modifica.

Acquista uno slot per dispositivo (AWS CLI)

Per acquistare l'offerta, è possibile eseguire il comando purchase-offering.

Per elencare le impostazioni del tuo account Device Farm, incluso il numero massimo di slot per dispositivi che puoi acquistare e il numero di minuti di prova gratuiti rimanenti, esegui il get-account-settings comando. L'output sarà simile al seguente:

```
{
    "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
        }
    }
}
```

Per visualizzare l'elenco delle offerte disponibili di slot per i dispositivi, eseguire il comando list-offerings. Verrà visualizzato un output simile al seguente:

```
{
```

```
    "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"
        }
    ]
}
```

Per elencare le promozioni delle offerte disponibili, esegui il list-offering-promotions comando.

> **Note**
>
> Questo comando restituisce solo le promozioni che non hai ancora acquistato. Quando acquisti uno o più slot offerti tramite una promozione, quella promozione non compare più nei risultati.

Verrà visualizzato un output simile al seguente:

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

Per visualizzare lo stato dell'offerta, esegui il comando get-offering-status. Verrà visualizzato un output simile al seguente:

```
{
    "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
        }
    }
```

```
}
```

Per questa funzionalità sono disponibili anche i list-offering-transactions comandi renew-offering and. Per ulteriori informazioni, consulta AWS CLI riferimento.

Acquista uno slot per dispositivo (API)

1. Chiama l'GetAccountSettingsoperazione per elencare le impostazioni del tuo account.
2. Chiama l'ListOfferingsoperazione per elencare le offerte di slot per dispositivi disponibili.
3. Chiama l'ListOfferingPromotionsoperazione per elencare le offerte e le promozioni disponibili.

   > **Note**
   >
   > Questo comando restituisce solo le promozioni che non hai ancora acquistato. Quando acquisti uno o più slot offerti tramite una promozione, quella promozione non compare più nei risultati.

4. Chiama l'PurchaseOfferingoperazione per acquistare un'offerta.
5. Chiama l'GetOfferingStatusoperazione per conoscere lo stato dell'offerta.

Per questa funzionalità sono disponibili anche i ListOfferingTransactionscomandi RenewOfferingand.

Per informazioni sull'utilizzo dell'API Device Farm, vedereAutomazione di Device Farm.

Annullamento dello slot di un dispositivo in Device Farm

È possibile annullare il numero di slot del dispositivo sia per i test automatici che per l'accesso remoto. Per istruzioni, consulta una delle seguenti sezioni. L'importo addebitato sul tuo account per il prossimo ciclo di fatturazione verrà elencato sotto il campo del periodo di fatturazione.

Per ulteriori informazioni sugli slot dei dispositivi, consulta. Acquisto di uno slot per dispositivi in Device Farm

Annullare lo slot di un dispositivo (console)

1. Accedere alla console Device Farm all'indirizzo https://console.aws.amazon.com/devicefarm.
2. Nel riquadro di navigazione, scegli Mobile Device Testing, quindi scegli Device slots.

3. Nella pagina Acquista e gestisci gli slot dei dispositivi, puoi ridurre il numero di slot per dispositivi sia per i test automatici che per l'accesso remoto diminuendo il valore indicato nella sezione Prossimo periodo di fatturazione. L'importo addebitato sul tuo account per il ciclo di fatturazione successivo verrà elencato sotto il campo Periodo di fatturazione.
4. Scegli Save (Salva). Apparirà una finestra di conferma della modifica. Controlla le informazioni, quindi scegli Conferma per completare la transazione.

Annullare lo slot di un dispositivo (AWS CLI)

Puoi eseguire il renew-offering comando per modificare la quantità di dispositivi per il ciclo di fatturazione successivo.

Annullare lo slot di un dispositivo (API)

Chiama l'RenewOfferingoperazione per modificare la quantità di dispositivi nel tuo account.

Concetti di AWS Device Farm

Device Farm è un servizio di test di app che puoi utilizzare per testare e interagire con le tue app Android, iOS e Web su telefoni e tablet fisici reali ospitati da Amazon Web Services (AWS).

Questa sezione descrive importanti concetti di Device Farm.

- Supporto per dispositivi in AWS Device Farm
- Ambienti di test in AWS Device Farm
- Esecuzioni
- App
- Report in AWS Device Farm
- Sessioni

Per ulteriori informazioni sui tipi di test supportati in Device Farm, vedereFramework di test e test integrati in AWS Device Farm.

Supporto per dispositivi in AWS Device Farm

Le seguenti sezioni forniscono informazioni sul supporto dei dispositivi in Device Farm.

Argomenti

- Dispositivi supportati
- Pool di dispositivi
- Dispositivi privati
- Branding del dispositivo
- Slot per dispositivi
- App preinstallate per dispositivi
- Funzionalità del dispositivo

Dispositivi supportati

Device Farm fornisce supporto per centinaia di dispositivi Android e iOS unici e popolari e combinazioni di sistemi operativi. L'elenco dei dispositivi disponibili cresce con l'immissione sul

mercato di nuovi dispositivi. Per l'elenco completo dei dispositivi, consulta l'elenco interattivo dei dispositivi nella AWS console.

Pool di dispositivi

Device Farm organizza i suoi dispositivi in pool di dispositivi che puoi utilizzare per i test. Questi pool di dispositivi contengono dispositivi correlati, ad esempio dispositivi che funzionano solo su Android o solo su iOS. Device Farm offre pool di dispositivi selezionati, come quelli per i dispositivi migliori. Puoi anche creare pool di dispositivi che uniscano dispositivi pubblici e privati.

Dispositivi privati

I dispositivi privati ti consentono di specificare le configurazioni hardware e software precise in base alle tue esigenze di test. Alcune configurazioni, come i dispositivi Android con root, possono essere supportate come dispositivi privati. Ogni dispositivo privato è un dispositivo fisico che Device Farm distribuisce per tuo conto in un data center Amazon. I tuoi dispositivi privati sono disponibili esclusivamente per te, sia per i test automatici sia per quelli manuali. Dopo aver deciso di terminare l'abbonamento, l'hardware verrà rimosso dal nostro ambiente. Per ulteriori informazioni, consulta Dispositivi privati e Dispositivi privati in AWS Device Farm.

Branding del dispositivo

Device Farm esegue test su dispositivi mobili e tablet fisici di una varietà di OEMs.

Slot per dispositivi

Gli slot per dispositivi corrispondono a simultaneità, dove il numero di slot per dispositivi acquistati determina il numero di dispositivi che puoi eseguire in sessioni di accesso da remoto o test.

Esistono due tipi di slot per dispositivi:

- Uno slot per dispositivi con accesso remoto può essere eseguito in sessioni di accesso remoto simultaneamente.

  Se hai uno slot per dispositivi ad accesso remoto, puoi solo eseguire una sessione di accesso remoto alla volta. Se acquisti ulteriori slot per dispositivi di test da remoto, puoi eseguire sessioni multiple simultaneamente.
- Uno slot per dispositivi di test automatizzati è uno slot sul quale puoi eseguire test simultaneamente.

Se hai uno slot per dispositivi di test automatizzati, puoi eseguire i test solo su un dispositivo alla volta. Se acquisti ulteriori slot per dispositivi di test automatizzati, puoi eseguire test multipli simultaneamente, su più dispositivi, per ottenere più velocemente i risultati dei test.

Puoi acquistare slot per dispositivi in base alla famiglia del dispositivo (dispositivi Android o iOS per test automatizzati e dispositivi Android e iOS per accesso remoto). Per ulteriori informazioni, consulta Prezzi di Device Farm.

App preinstallate per dispositivi

I dispositivi in Device Farm includono un numero limitato di app già installate da produttori e gestori.

Funzionalità del dispositivo

Tutti i dispositivi dispongono di connettività Internet. I dispositivi non sono connessi con alcun operatore e non possono effettuare chiamate né inviare SMS.

Puoi scattare fotografie con qualsiasi dispositivo che supporta una telecamera frontale o posteriore. A causa della modalità di montaggio dei dispositivi, le foto potrebbero apparire scure e sfocate.

Google Play Services e Google Chrome sono installati su dispositivi Android.

Ambienti di test in AWS Device Farm

AWS Device Farm fornisce ambienti di test personalizzati e standard per eseguire test automatici. Puoi scegliere un ambiente di test personalizzato per ottenere il controllo completo dei test automatizzati. In alternativa, puoi scegliere l'ambiente di test standard predefinito di Device Farm, che offre report granulari di ogni test nella tua suite di test automatizzata.

Argomenti

Ambiente di test standard

Quando esegui un test nell'ambiente standard, Device Farm fornisce log e report dettagliati per ogni caso della tua suite di test. Puoi visualizzare dati sulle prestazioni, video, screenshot e log per ciascun test per individuare e risolvere problemi all'interno dell'app.

> **Note**
>
> Poiché Device Farm fornisce report granulari nell'ambiente standard, i tempi di esecuzione dei test possono essere più lunghi rispetto a quelli eseguiti localmente. Se desideri tempi più rapidi, esegui i test in un ambiente personalizzato.

Ambiente di test personalizzato

Quando personalizzi l'ambiente di test, puoi specificare i comandi che Device Farm deve eseguire per eseguire i test. Ciò garantisce che i test su Device Farm vengano eseguiti in modo simile ai test eseguiti sul computer locale. Eseguire i test in questa modalità consente inoltre la creazione di log e lo streaming del video in tempo reale dei tuoi test. Quando esegui test in un ambiente personalizzato, non ottieni report granulari per ciascun caso di test. Per ulteriori informazioni, consulta Ambienti di test personalizzati in AWS Device Farm.

È possibile utilizzare un ambiente di test personalizzato quando si utilizza la console Device Farm o l'API Device Farm per creare un'esecuzione di test. AWS CLI

Per ulteriori informazioni, consulta Caricamento di una specifica di test personalizzata utilizzando and. AWS CLICreazione di un'esecuzione di test in Device Farm

Funziona in AWS Device Farm

Le seguenti sezioni contengono informazioni sulle esecuzioni in Device Farm.

Un'esecuzione in Device Farm rappresenta una build specifica dell'app, con una serie specifica di test, da eseguire su un set specifico di dispositivi. Un'esecuzione produce un rapporto che contiene informazioni sui risultati dell'esecuzione. Un'esecuzione contiene uno o più processi.

Argomenti

- Esegui la configurazione
- Esegui la conservazione dei file
- Esegui lo stato del dispositivo
- Esecuzioni parallele
- Impostazione del timeout di esecuzione
- Annunci nelle puntate

- File multimediali in tirature
- Attività comuni per le esecuzioni

Esegui la configurazione

Come parte di un'esecuzione, è possibile fornire impostazioni che Device Farm può utilizzare per sovrascrivere le impostazioni correnti del dispositivo. Queste includono coordinate di latitudine e longitudine, dati aggiuntivi (contenuti in un file.zip) e app ausiliarie (app che devono essere installate prima dell'app da testare). Su Android, è possibile modificare alcune impostazioni aggiuntive, come lo stato locale e della radio (Bluetooth, GPS, NFC e Wi-Fi).

Esegui la conservazione dei file

Device Farm archivia le app e i file per 30 giorni, quindi li elimina dal sistema. Tuttavia, puoi eliminare i tuoi file in qualsiasi momento.

Device Farm archivia i risultati delle corse, i log e gli screenshot per 400 giorni, quindi li elimina dal sistema.

Esegui lo stato del dispositivo

Device Farm riavvia sempre un dispositivo prima di renderlo disponibile per il lavoro successivo.

Esecuzioni parallele

Device Farm esegue test in parallelo non appena i dispositivi diventano disponibili.

Impostazione del timeout di esecuzione

È possibile impostare un valore per la durata della sessione di un test prima di arrestare l'esecuzione del test su ogni dispositivo. Ad esempio, se il completamento dei tuoi test richiede 20 minuti per dispositivo, è opportuno scegliere un timeout di 30 minuti per dispositivo.

Per ulteriori informazioni, consulta Impostazione del timeout di esecuzione per le esecuzioni di test in AWS Device Farm.

Annunci nelle puntate

Ti consigliamo di rimuovere gli annunci dalle tue app prima di caricarli su Device Farm. Non è garantito che gli annunci vengano visualizzati durante le esecuzioni.

File multimediali in tirature

Puoi fornire contenuti multimediali o altri dati per accompagnare la tua app. I dati aggiuntivi devono essere forniti in un file .zip delle dimensioni massime di 4 GB.

Attività comuni per le esecuzioni

Per ulteriori informazioni, consultare Creazione di un'esecuzione di test in Device Farm e I test vengono eseguiti in AWS Device Farm.

App in AWS Device Farm

Le seguenti sezioni contengono informazioni sui comportamenti delle app in Device Farm.

Argomenti

- App di strumentazione
- Riassegnare la firma delle app durante le esecuzioni
- App offuscate in corso di esecuzione

App di strumentazione

Non è necessario strumentare le app o fornire a Device Farm il codice sorgente delle app. È possibile inviare app Android non modificate. Le app iOS devono essere compilate con il dispositivo iOS target anziché con il simulatore.

Riassegnare la firma delle app durante le esecuzioni

Per le app iOS, non è necessario aggiungere alcun Device Farm UUIDs al profilo di provisioning. Device Farm sostituisce il profilo di provisioning incorporato con un profilo wildcard e quindi firma nuovamente l'app. Se fornisci dati ausiliari, Device Farm li aggiunge al pacchetto dell'app prima che Device Farm li installi, in modo che l'ausiliario esista nella sandbox dell'app. La nuova firma dell'app rimuove diritti come App Group, Associated Domains, Game Center,, Wireless Accessory Configuration HealthKit, In-App Purchase HomeKit, Inter-App Audio, Apple Pay, Notifiche push e Configurazione e controllo VPN.

Per le app Android, Device Farm firma nuovamente l'app. Ciò potrebbe interrompere qualsiasi funzionalità che dipende dalla firma dell'app, come l'API Android di Google Maps, oppure potrebbe attivare il rilevamento antipirateria o antimanomissione da parte di prodotti come. DexGuard

App offuscate in corso di esecuzione

Per le app Android, se l'app è offuscata, puoi comunque testarla con Device Farm se la usi. ProGuard Tuttavia, se utilizzi misure DexGuard antipirateria, Device Farm non può firmare nuovamente ed eseguire test sull'app.

Report in AWS Device Farm

Le seguenti sezioni forniscono informazioni sui report dei test di Device Farm.

Argomenti

Conservazione dei report

Device Farm archivia i tuoi report per 400 giorni. Questi report includono metadati, log, screenshot e dati sulle prestazioni.

Componenti dei report

I report in Device Farm contengono informazioni su passaggi e fallimenti, segnalazioni di arresti anomali, registri di test e dispositivi, schermate e dati sulle prestazioni.

I report includono inoltre dati approfonditi e risultati generali per dispositivo, ad esempio il numero di occorrenze di un determinato problema.

Registra i report di accesso

I report contengono tutti i logcat acquisiti per i test Android e i log completi della console del dispositivo per i test iOS.

Attività comuni per i report

Per ulteriori informazioni, consulta Visualizzazione dei report dei test in Device Farm.

Sessioni in AWS Device Farm

Puoi utilizzare Device Farm per eseguire test interattivi delle app Android e iOS tramite sessioni di accesso remoto. Ciò include sia l'interazione manuale in un browser Web sia l'esecuzione dei test Appium da un client locale sul dispositivo remoto. Gli sviluppatori possono riprodurre i problemi con la loro app o con i test Appium su un dispositivo specifico per isolare e risolvere i problemi.

Argomenti

Dispositivi supportati per l'accesso remoto

Device Farm fornisce supporto per una serie di dispositivi Android e iOS unici e popolari. L'elenco dei dispositivi disponibili cresce con l'immissione sul mercato di nuovi dispositivi. La console Device Farm mostra l'elenco corrente dei dispositivi Android e iOS disponibili per l'accesso remoto. Per ulteriori informazioni, consulta Supporto per dispositivi in AWS Device Farm.

Conservazione dei file di sessione

Device Farm archivia le app e i file per 30 giorni, quindi li elimina dal sistema. Tuttavia, puoi eliminare i tuoi file in qualsiasi momento.

Device Farm archivia i registri delle sessioni e i video acquisiti per 400 giorni, quindi li elimina dal sistema.

App di strumentazione

Non è necessario strumentare le app o fornire a Device Farm il codice sorgente delle app. Le app Android e iOS possono essere inviate senza modifiche.

Riassegnare la firma delle app nelle sessioni

Device Farm rifirma le app Android e iOS. Ciò può causare l'interruzione della funzionalità che dipende dalla firma dell'app. Ad esempio, l'API di Google Maps per Android dipende dalla firma

della tua app. La nuova firma delle app può inoltre attivare il rilevamento delle norme antipirateria o antimanomissione da parte di prodotti come quelli per dispositivi Android. DexGuard

App offuscate nelle sessioni

Per le app Android, se l'app è offuscata, puoi comunque testarla con Device Farm se la usi. ProGuard Tuttavia, se utilizzi misure DexGuard antipirateria, Device Farm non può firmare nuovamente l'app.

Progetti in AWS Device Farm

Un progetto in Device Farm rappresenta uno spazio di lavoro logico in Device Farm che contiene esecuzioni, una esecuzione per ogni test di una singola app su uno o più dispositivi. I progetti ti consentono di organizzare gli spazi di lavoro nel modo che preferisci. Ad esempio, può esserci un progetto per titolo dell'app o un progetto per piattaforma. È possibile creare tutti i progetti necessari.

Puoi utilizzare la console AWS Device Farm, AWS Command Line Interface (AWS CLI) o l'API AWS Device Farm per lavorare con progetti.

Argomenti

Creazione di un progetto in AWS Device Farm

Puoi creare un progetto utilizzando la console AWS Device Farm o l' AWS CLI API AWS Device Farm.

Prerequisiti

- Completa le fasi descritte in Configurazione.

Crea un progetto (console)

1. Accedere alla console Device Farm all'indirizzo https://console.aws.amazon.com/devicefarm.
2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.
3. Scegliere New project (Nuovo progetto).
4. Inserire un nome per il progetto. Facoltativamente, puoi fornire uno o più dei parametri seguenti, quindi scegliere Invia.

   Impostazioni Virtual Private Cloud (VPC)

   Seleziona un VPC, sottoreti e gruppo di sicurezza da applicare al dispositivo in test e al relativo host di test associato. Questa funzionalità è supportata solo con dispositivi privati. Per ulteriori informazioni, consulta VPC-ENI nella Device Farm di AWS.

Ruolo di esecuzione ARN

Un ruolo IAM che deve essere assunto dal test runner in ambienti di test personalizzati. Per ulteriori informazioni, consulta Accedi alle risorse AWS utilizzando un ruolo di esecuzione IAM.

Variabili di ambiente

Una o più variabili da inserire nell'ambiente del processo di esecuzione del test. I nomi delle variabili che iniziano con «DEVICEFARM_» sono riservati all'uso del servizio. Sconsigliamo di archiviare valori sensibili in queste variabili di ambiente e suggeriamo invece di utilizzare un ruolo di esecuzione IAM per recuperare tali valori da AWS Secrets Manager durante il test.

Crea un progetto ()AWS CLI

- Eseguire create-project specificando il nome del progetto.

  Esempio:

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

  La AWS CLI risposta include l'Amazon Resource Name (ARN) del progetto.

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

  Per ulteriori informazioni, consultare create-project e AWS CLI riferimento.

Crea un progetto (API)

- Chiamata dell'API `CreateProject`.

Per informazioni sull'utilizzo dell'API Device Farm, vedereAutomazione di Device Farm.

Visualizzazione dell'elenco dei progetti in AWS Device Farm

Puoi utilizzare la console AWS Device Farm o AWS CLI l'API AWS Device Farm per visualizzare l'elenco dei progetti.

Argomenti

Prerequisiti

- Crea almeno un progetto in Device Farm. Segui le istruzioni riportate in Creazione di un progetto in AWS Device Farm, poi torna a questa pagina.

Visualizza l'elenco dei progetti (console)

1. Accedere alla console Device Farm all'indirizzo https://console.aws.amazon.com/devicefarm.
2. Per trovare l'elenco dei progetti disponibili, procedi come segue:
   - Per i progetti di test dei dispositivi mobili, nel menu di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.
   - Per i progetti di test dei browser desktop, nel menu di navigazione di Device Farm, scegli Desktop Browser Testing, quindi scegli Progetti.

Visualizza l'elenco dei progetti (AWS CLI)

- Per visualizzare l'elenco dei progetti, esegui il comando list-projects.

  Per visualizzare informazioni su un singolo progetto, esegui il comando get-project.

Per informazioni sull'utilizzo di Device Farm con AWS CLI, vedereAWS CLI riferimento.

Visualizza l'elenco dei progetti (API)

- Per visualizzare l'elenco dei progetti, chiama l'API `ListProjects`.

  Per visualizzare informazioni su un singolo progetto, chiama l'API `GetProject`.

Per informazioni sull'API AWS Device Farm, consultaAutomazione di Device Farm.

I test vengono eseguiti in AWS Device Farm

Un'esecuzione in Device Farm rappresenta una build specifica dell'app, con una serie specifica di test, da eseguire su un set specifico di dispositivi. Un'esecuzione produce un rapporto che contiene informazioni sui risultati dell'esecuzione. Un'esecuzione contiene uno o più processi. Per ulteriori informazioni, consulta Esecuzioni.

Puoi utilizzare la console AWS Device Farm, AWS Command Line Interface (AWS CLI) o l'API AWS Device Farm per lavorare con le esecuzioni di test.

Argomenti

Creazione di un'esecuzione di test in Device Farm

Puoi utilizzare la console Device Farm o AWS CLI l'API Device Farm per creare un'esecuzione di test. Puoi anche utilizzare un plug-in supportato, come i plugin Jenkins o Gradle per Device Farm. Per ulteriori informazioni sui plugin , consulta Strumenti e plugin. Per informazioni sulle sessioni, consulta Esecuzioni.

Argomenti

Prerequisiti

È necessario disporre di un progetto in Device Farm. Segui le istruzioni riportate in Creazione di un progetto in AWS Device Farm, poi torna a questa pagina.

Crea un'esecuzione di test (console)

1. Accedere alla console Device Farm all'indirizzo https://console.aws.amazon.com/devicefarm.
2. Nel riquadro di navigazione, scegli Mobile Device Testing, quindi scegli Progetti.
3. Se è già stato creato un progetto, è possibile caricare i test. Altrimenti, scegliete Nuovo progetto, inserite un nome di progetto, quindi scegliete Crea.
4. Apri il progetto, quindi scegli Crea esegui.
5. (Facoltativo) In Impostazioni di esecuzione, nella sezione Nome esecuzione, inserisci un nome per la corsa. Se non viene fornito alcun nome, la console Device Farm chiamerà l'esecuzione «My Device Farm run» per impostazione predefinita.
6. (Facoltativo) In Impostazioni di esecuzione, nella sezione Job timeout, puoi specificare il timeout di esecuzione per l'esecuzione del test. Se utilizzi slot di test illimitati, conferma che Unmetered sia selezionato in Metodo di fatturazione.
7. In Impostazioni di corsa, nella sezione Tipo di corsa, seleziona il tipo di corsa. Seleziona l'app per Android se non hai un'app pronta per il test o se stai testando un'app per Android (.apk). Seleziona App iOS se stai testando un'app iOS (.ipa). Seleziona App Web se desideri testare le applicazioni Web.
8. In Seleziona app, nella sezione Opzioni di selezione app, scegli Seleziona app di esempio fornita da Device Farm se non hai un'app disponibile per il test. Se porti la tua app, seleziona Carica la tua app e scegli il file dell'applicazione. Se si sta caricando un'app iOS, assicurarsi di selezionare iOS device (dispositivo iOS), invece di un simulatore.
9. In Configure test, scegli uno dei framework di test disponibili.

> **Note**
>
> Se non si dispone di alcun test disponibile, selezionare Built-in: Fuzz (Integrata: Fuzz) per eseguire una suite di test integrata standard. Se si sceglie Built-in: Fuzz (Integrata: Fuzz) e vengono visualizzate le caselle Event count (Conteggio eventi), Event throttle (Limite eventi) e Randomizer seed (Seed randomizer), è possibile modificare o mantenere i valori.

Per informazioni sulle suite di test disponibili, consulta Framework di test e test integrati in AWS Device Farm.

10. Se non hai scelto Built-in: Fuzz, seleziona Scegli file in Seleziona pacchetto di test. Cerca e scegli il file che contiene i tuoi test.
11. Per il tuo ambiente di test, scegli Esegui il test nel nostro ambiente standard o Esegui il test in un ambiente personalizzato. Per ulteriori informazioni, consulta Ambienti di test in AWS Device Farm.
12. Se utilizzi un ambiente di test personalizzato, puoi facoltativamente fare quanto segue:
    - Se si desidera modificare il file di specifica predefinito in un ambiente di test personalizzato, selezionare Edit (Modifica) per aggiornare la specifica YAML predefinita.
    - Se hai modificato le specifiche del test, scegli Salva come nuovo per aggiornarle.
    - È possibile configurare le variabili di ambiente. Le variabili fornite qui avranno la precedenza su quelle che possono essere configurate nel progetto principale.
13. In Seleziona dispositivi, effettuate una delle seguenti operazioni:
    - Per selezionare un pool di dispositivi integrato per eseguire i test, per Device pool (Pool di dispositivi), selezionare Top Devices (Dispositivi migliori).
    - Per creare il pool di dispositivi per eseguire i test, segui le istruzioni contenute in Creazione di un pool di dispositivi, quindi torna a questa pagina.
    - Se hai creato il pool di dispositivi in precedenza, per Device pool (Pool di dispositivi), selezionare il pool di dispositivi.
    - Seleziona Seleziona manualmente i dispositivi e scegli i dispositivi desiderati su cui vuoi correre. Questa configurazione non verrà salvata.

    Per ulteriori informazioni, consulta Supporto per dispositivi in AWS Device Farm.
14. (Facoltativo) Per aggiungere una configurazione aggiuntiva, apri il menu a discesa Configurazione aggiuntiva. In questa sezione, puoi effettuare una delle seguenti operazioni:
    - Per fornire un ruolo di esecuzione ARN o sovrascrivere uno configurato nel progetto principale, utilizzare il campo ARN del ruolo di esecuzione.
    - Per fornire altri dati da utilizzare a Device Farm durante l'esecuzione, accanto a Aggiungi dati aggiuntivi, scegli Scegli file, quindi cerca e scegli il file.zip che contiene i dati.

- Per installare un'app aggiuntiva da utilizzare in Device Farm durante l'esecuzione, accanto a Installa altre app, scegli Scegli file, quindi cerca e scegli il file.apk o .ipa che contiene l'app. Ripetere questa operazione per altre applicazioni che si desidera installare. Puoi modificare l'ordine di installazione trascinando le app dopo averle caricate.
- Per specificare se una rete Wi-Fi, Bluetooth, GPS o NFC sia abilitata durante la sessione, accanto a Set radio states (Imposta stati radio), selezionare le caselle appropriate.
- Per preconfigurare la latitudine e la longitudine del dispositivo per la sessione, accanto a Device location (Posizione dispositivo), immettere le coordinate.
- Per preimpostare le impostazioni locali del dispositivo per l'esecuzione, in Impostazioni locali del dispositivo, scegli le impostazioni locali.
- Seleziona Abilita la registrazione video per registrare video durante il test.
- Seleziona Abilita l'acquisizione dei dati sulle prestazioni dell'app per acquisire i dati sulle prestazioni dal dispositivo.

> **Note**
>
> L'impostazione dello stato radio e delle impostazioni locali del dispositivo sono opzioni disponibili solo per i test nativi Android al momento.

> **Note**
>
> Se disponi di dispositivi privati, viene visualizzata anche la configurazione specifica per i dispositivi privati.

15. Nella parte inferiore della pagina, scegli Crea corsa per pianificare la corsa.

Device Farm avvia l'esecuzione non appena i dispositivi sono disponibili, in genere entro pochi minuti. Durante l'esecuzione del test, la console Device Farm visualizza un'icona

in sospeso nella tabella di esecuzione. Ogni dispositivo in esecuzione inizierà anche con l'icona in sospeso, quindi passerà all'icona

in esecuzione all'inizio del test. Al termine di ogni test, accanto al nome del dispositivo viene

visualizzata l'icona del risultato del test. Una volta completati tutti i test, l'icona in sospeso accanto all'esecuzione diventa l'icona del risultato del test.

Se desideri interrompere l'esecuzione del test, consultaInterruzione di un'esecuzione in AWS Device Farm.

Crea un test run (AWS CLI)

È possibile utilizzare il AWS CLI per creare un'esecuzione di test.

Argomenti

- Fase 1: Scegli un progetto
- Passaggio 2: Scegli un pool di dispositivi
- Passaggio 3: carica il file della tua candidatura
- Passaggio 4: carica il pacchetto di script di test
- Fase 5: (Facoltativo) Carica le specifiche di test personalizzate
- Passaggio 6: Pianifica un'esecuzione di test

Fase 1: Scegli un progetto

È necessario associare l'esecuzione del test a un progetto Device Farm.

1. Per elencare i tuoi progetti Device Farm, eseguilist-projects. Se non disponi ancora di un progetto, consulta Creazione di un progetto in AWS Device Farm.

   Esempio:

   ```
   aws devicefarm list-projects
   ```

   La risposta include un elenco dei tuoi progetti Device Farm.

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

```
            "created": 1503612890.057
        }
    ]
}
```

2. Scegliere un progetto da associare alla sessione di test e prendere nota del suo Amazon Resource Name (ARN).

Passaggio 2: Scegli un pool di dispositivi

È necessario selezionare un pool di dispositivi da associare alla sessione di test.

1. Per visualizzare i pool di dispositivi, eseguire list-device-pools specificando l'ARN del progetto.

   Esempio:

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

   La risposta include i pool di dispositivi Device Farm integrati, ad esempioTop Devices, e tutti i pool di dispositivi creati in precedenza per questo progetto:

   ```
   {
       "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. Scegliere un pool di dispositivi e prendere nota del suo ARN.

   È inoltre possibile creare un pool di dispositivi e ritornare a questa fase. Per ulteriori informazioni, consulta Crea un pool di dispositivi ()AWS CLI.

Passaggio 3: carica il file della tua candidatura

Per creare la tua richiesta di caricamento e ottenere un URL di caricamento predefinito di Amazon Simple Storage Service (Amazon S3), devi:

- L'ARN di progetto.
- Il nome del file di applicazione.
- Il tipo di caricamento.

Per ulteriori informazioni, consulta create-upload.

1. Per caricare un file, eseguire create-upload con i parametri `--project-arn`, `--type` e `--name`.

   Questo esempio crea un caricamento per un'applicazione Android:

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

   La risposta include l'ARN di caricamento dell'applicazione e un URL prefirmato.

   ```
   {
       "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. Prendere nota dell'ARN di caricamento dell'applicazione e dell'URL prefirmato.
3. Carica il file dell'app utilizzando l'URL predefinito di Amazon S3. Questo esempio utilizza curl per caricare un file .apk Android:

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

   Per ulteriori informazioni, consulta Caricamento di oggetti utilizzando presigned URLs nella Guida per l'utente di Amazon Simple Storage Service.

4. Per verificare lo stato del caricamento dell'applicazione, eseguire get-upload e specificare l'ARN di caricamento dell'applicazione.

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

   Attendere che lo stato della risposta sia SUCCEEDED prima di caricare il pacchetto degli script di test.

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

Passaggio 4: carica il pacchetto di script di test

Quindi, caricare il pacchetto degli script di test.

1. Per creare la tua richiesta di caricamento e ottenere un URL di caricamento predefinito per Amazon S3, esegui create-upload con i parametri `--project-arn`--`name`, and. `--type`

   Questo esempio crea un caricamento del pacchetto di test Appium Java TestNG:

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

   La risposta include l'ARN di caricamento del pacchetto di test e un URL prefirmato.

   ```
   {
       "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. Prendere nota dell'ARN di caricamento del pacchetto di test e dell'URL prefirmato.
3. Carica il file del pacchetto degli script di test utilizzando l'URL predefinito di Amazon S3. Questo esempio utilizza curl per caricare un file .zip degli script Appium TestNG.

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

4. Per verificare lo stato del caricamento del pacchetto degli script di test, eseguire get-upload e specificare l'ARN di caricamento del pacchetto di test dalla fase 1.

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

Attendere che lo stato della risposta sia SUCCEEDED prima di procedere alla fase successiva facoltativa.

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

Fase 5: (Facoltativo) Carica le specifiche di test personalizzate

Se si eseguono i test in un ambiente di test standard, saltare questa fase.

Device Farm mantiene un file delle specifiche di test predefinito per ogni tipo di test supportato. Quindi, scaricare la specifica di test predefinita e utilizzarla per creare un caricamento della specifica personalizzata per l'esecuzione dei test in un ambiente di test personalizzato. Per ulteriori informazioni, consulta Ambienti di test in AWS Device Farm.

1. Per trovare l'ARN di caricamento per la specifica di test predefinita, eseguire list-uploads e specificare l'ARN del progetto.

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

La risposta contiene una voce per ciascuna specifica di test predefinita:

```
{
    "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. Scegliere la specifica di test predefinita dall'elenco. Prendere nota del suo ARN di caricamento.
3. Per scaricare la specifica di test predefinita, eseguire get-upload e specificare l'ARN di caricamento.

   Esempio:

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

   La risposta contiene un URL prefirmato dove è possibile scaricare la specifica di test predefinita.
4. Questo esempio utilizza curl per scaricare la specifica di test predefinita e salvarla come `MyTestSpec.yml`:

   ```
   curl "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/ExampleURL" >
    MyTestSpec.yml
   ```
5. Puoi modificare la specifica di test predefinita per soddisfare i requisiti di test e utilizzare la specifica di test modificata in sessioni di test future. Saltare questa fase per utilizzare la specifica di test predefinita così come avviene in un ambiente di test personalizzato.
6. Per creare un caricamento della specifica di test personalizzata, eseguire create-upload specificando il nome della specifica di test, il tipo della specifica di test e l'ARN di progetto.

   Questo esempio crea un caricamento per una specifica di test personalizzata Appium Java TestNG:

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

   La risposta include l'ARN di caricamento della specifica di test e un URL prefirmato:

```
{
    "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. Prendere nota dell'ARN per il caricamento della specifica di test e dell'URL prefirmato.
8. Carica il file delle specifiche di test utilizzando l'URL predefinito di Amazon S3. Questo esempio utilizza curl per caricare una specifica di test JavaTest Appium NG:

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

9. Per verificare lo stato del caricamento della specifica di test, eseguire get-upload e specificare l'ARN di caricamento.

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

Attendere che lo stato della risposta sia SUCCEEDED prima di pianificare la sessione di test.

```
{
    "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}"
    }
```

```
}
```

Per aggiornare la specifica di test personalizzata, eseguire update-upload specificando l'ARN di caricamento per la specifica di test. Per ulteriori informazioni, consulta update-upload.

Passaggio 6: Pianifica un'esecuzione di test

Per pianificare un'esecuzione di test con AWS CLI, eseguischedule-run, specificando:

- L'ARN del progetto dalla fase 1.
- L'ARN del pool dei dispositivi dalla fase 2.
- L'ARN di caricamento dell'applicazione dalla fase 3.
- L'ARN di caricamento del pacchetto di test dalla fase 4.

Se si eseguono i test in un ambiente di test personalizzato, è necessario anche l'ARN della specifica di test dalla fase 5.

Per pianificare una sessione in un ambiente di test standard

- Eseguire schedule-run specificando l'ARN di progetto, l'ARN del pool di dispositivi, l'ARN di caricamento dell'applicazione e le informazioni del pacchetto di test.

  Esempio:

  ```
  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
  ```

  La risposta contiene un ARN di sessione che puoi utilizzare per verificare lo stato della sessione di test.

  ```
  {
      "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
        }
    }
}
```

Per ulteriori informazioni, consulta schedule-run.

Per pianificare una sessione in un ambiente di test personalizzato

- Le fasi sono quasi uguali a quelle per l'ambiente di test standard con un attributo aggiuntivo `testSpecArn` nel parametro `--test`.

  Esempio:

```
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
```

Per verificare lo stato della sessione di test

- Utilizzare il comando get-run e specificare l'ARN di sessione:

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

Per ulteriori informazioni, consulta get-run. Per informazioni sull'utilizzo di Device Farm con AWS CLI, vedereAWS CLI riferimento.

Creare un test run (API)

I passaggi sono gli stessi descritti nella AWS CLI sezione. Per informazioni, consulta Crea un test run (AWS CLI).

Queste informazioni sono necessarie per chiamare l'API `ScheduleRun`:

- Un ARN di progetto. Consulta Crea un progetto (API) e `CreateProject`.
- Un ARN di caricamento dell'applicazione. Per informazioni, consulta `CreateUpload`.
- Un ARN di caricamento del pacchetto di test. Per informazioni, consulta `CreateUpload`.
- Un ARN del pool di dispositivi. Consulta Creazione di un pool di dispositivi e `CreateDevicePool`.

> **Note**
>
> Se si eseguono i test in un ambiente di test personalizzato, è necessario anche l'ARN di caricamento della specifica di test. Per ulteriori informazioni, consultare Fase 5: (Facoltativo) Carica le specifiche di test personalizzate e `CreateUpload`.

Per informazioni sull'utilizzo dell'API Device Farm, vedereAutomazione di Device Farm.

Fasi successive

Nella console Device Farm, l'icona dell'orologio

diventa un'icona del risultato, ad esempio successo al

termine dell'esecuzione. Al termine dei test, viene visualizzato un rapporto per la sessione. Per ulteriori informazioni, consulta Report in AWS Device Farm.

Per utilizzare il rapporto, segui le istruzioni riportate in Visualizzazione dei report dei test in Device Farm.

Impostazione del timeout di esecuzione per le esecuzioni di test in AWS Device Farm

È possibile impostare un valore per la durata della sessione di un test prima di arrestare l'esecuzione del test su ogni dispositivo. Il timeout di esecuzione predefinito è di 150 minuti per dispositivo, ma è possibile impostare un valore minimo di 5 minuti. Puoi utilizzare la console AWS Device Farm o l'API AWS Device Farm per impostare il timeout di esecuzione. AWS CLI

> **Important**
>
> Il timeout di esecuzione deve essere impostato per la durata massima di una sessione di test, insieme ad alcuni buffer. Ad esempio, se i tuoi test durano 20 minuti per dispositivo, è opportuno scegliere un timeout di 30 minuti per dispositivo.

Se l'esecuzione supera il timeout, l'esecuzione su quel dispositivo è arrestata forzatamente. Sono disponibili i risultati parziali, laddove possibile. Se si utilizza l'opzione di fatturazione con misurazione, l'esecuzione è fatturata fino a quel punto. Per ulteriori informazioni sui prezzi, consulta la pagina dei prezzi di Device Farm.

È possibile utilizzare questa funzione se si conosce la possibile durata di una sessione di test su ciascun dispositivo. Quando si specifica un timeout di esecuzione per una sessione di test, è possibile evitare che una sessione di test sia bloccata per qualsiasi motivo e che vengano fatturati minuti in cui i test non erano in esecuzione sul dispositivo. In altre parole, utilizzando la funzione del timeout di esecuzione è possibile arrestare una sessione nel caso in cui stia durando più del previsto.

È possibile impostare il timeout di esecuzione in due modi: a livello del progetto e a livello della sessione di test.

Prerequisiti

1. Completa le fasi descritte in Configurazione.
2. Crea un progetto in Device Farm. Segui le istruzioni riportate in Creazione di un progetto in AWS Device Farm, poi torna a questa pagina.

Imposta il timeout di esecuzione per un progetto

1. Accedere alla console Device Farm all'indirizzo https://console.aws.amazon.com/devicefarm.
2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.
3. Se hai già un progetto, selezionalo dall'elenco. Altrimenti, scegli Nuovo progetto, inserisci un nome per il progetto, quindi scegli Invia.
4. Selezionare Project settings (Impostazioni del progetto).
5. Nella scheda General (Generale) per Execution timeout (Timeout di esecuzione), immettere un valore oppure utilizzare la barra di scorrimento.
6. Scegli Save (Salva).

   Ora tutte le sessioni di test nel proprio progetto utilizzano il valore del timeout di esecuzione, a meno che non si sovrascriva il valore del timeout quando si pianifica una sessione.

Imposta il timeout di esecuzione per un'esecuzione di test

1. Accedere alla console Device Farm all'indirizzo https://console.aws.amazon.com/devicefarm.
2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.
3. Se hai già un progetto, selezionalo dall'elenco. Altrimenti, scegli Nuovo progetto, inserisci un nome per il progetto, quindi scegli Invia.
4. Scegliere Create a new run (Crea una nuova sessione).
5. Seguire la procedura per scegliere un'applicazione, configurare il test, selezionare i propri dispositivi e specificare lo stato dei dispositivi.
6. In Review and start run, per Imposta il timeout di esecuzione, inserisci un valore o usa la barra di scorrimento.

7. Selezionare Confirm and start run (Controlla e avvia sessione).

Simulazione delle connessioni e delle condizioni di rete per le esecuzioni di AWS Device Farm

Puoi utilizzare il network shaping per simulare le connessioni e le condizioni di rete durante il test delle tue app Android, iOS e web in Device Farm. Ad esempio, puoi simulare una connettività Internet con perdite o intermittente.

Quando si crea una sessione utilizzando le impostazioni di rete predefinite, ogni dispositivo dispone di una connessione Wi-Fi completa e gratuita con connettività Internet. Quando si utilizza il network shaping, è possibile modificare la connessione Wi-Fi per specificare un profilo di rete come 3G o Lossy WiFi che controlli la velocità effettiva, il ritardo, il jitter e la perdita per il traffico in entrata e in uscita.

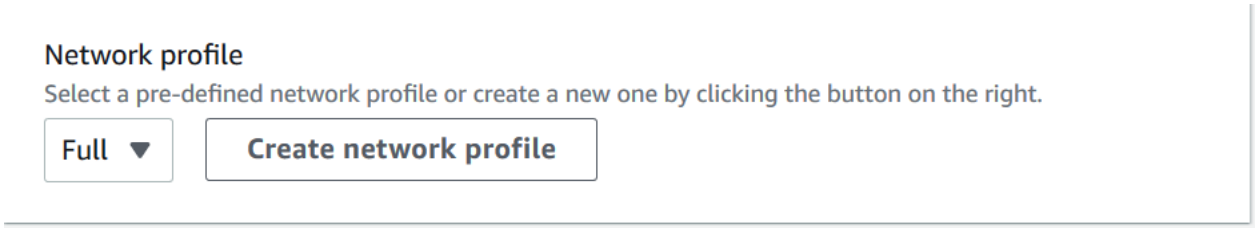
Argomenti

Imposta la configurazione della rete quando pianifichi un'esecuzione di test

Quando pianifichi una corsa, puoi scegliere uno qualsiasi dei profili curati da Device Farm oppure puoi crearne e gestirne uno tuo.

1. Da qualsiasi progetto Device Farm, scegli Crea una nuova esecuzione.

   Se non hai ancora un progetto, consulta Creazione di un progetto in AWS Device Farm.
2. Scegli l'applicazione, quindi scegli Avanti.
3. Configura il test, quindi scegli Avanti.
4. Seleziona i tuoi dispositivi, quindi scegli Avanti.
5. Nella sezione Impostazioni di posizione e rete, scegli un profilo di rete o scegli Crea profilo di rete per crearne uno personalizzato.

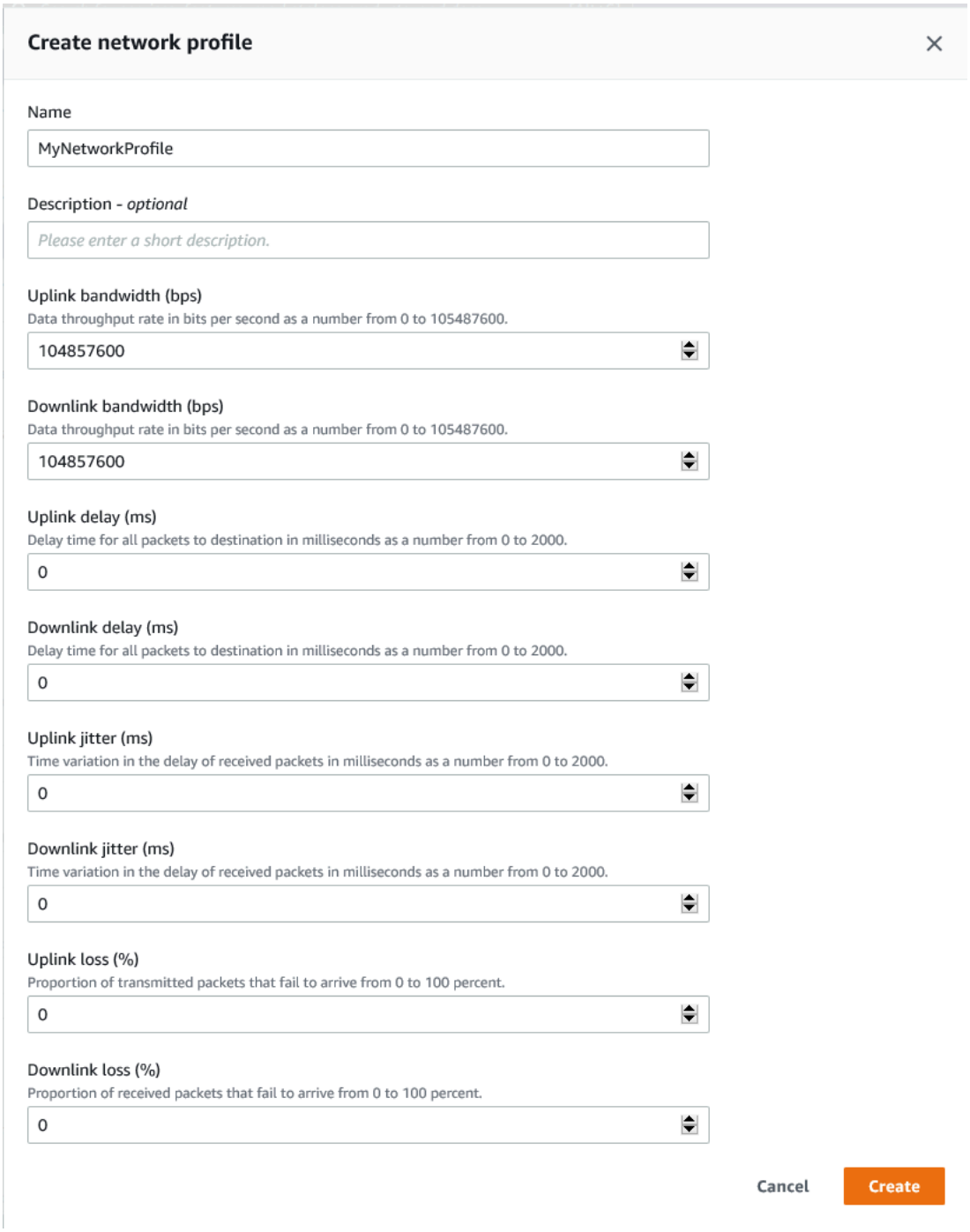


6. Scegli Next (Successivo).
7. Verifica e avvio della sessione di test.

Crea un profilo di rete

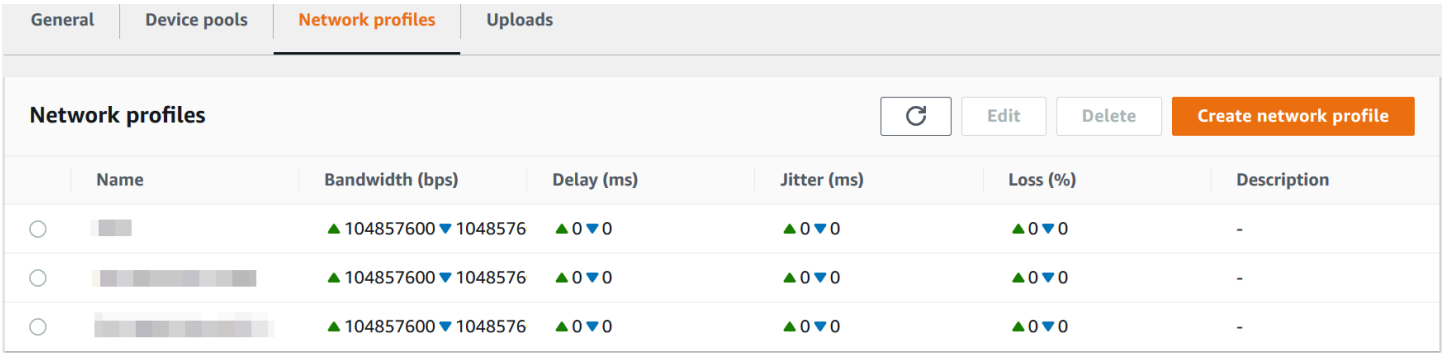
Quando si crea una sessione di test, è possibile creare un profilo di rete.

1. Scegli Crea profilo di rete.



2. Immettere un nome e le impostazioni per il proprio profilo di rete.
3. Scegli Create (Crea).
4. Termina di creare la tua sessione di test e avvia l'esecuzione.

Dopo aver creato un profilo di rete, potrai visualizzarlo e gestirlo nella pagina Project settings (Impostazioni del progetto).



Modifica le condizioni della rete durante il test

Puoi chiamare un'API dall'host del tuo dispositivo utilizzando un framework come Appium per simulare condizioni di rete dinamiche come una larghezza di banda ridotta durante l'esecuzione del test. Per ulteriori informazioni, consulta CreateNetworkProfile.

Interruzione di un'esecuzione in AWS Device Farm

È possibile arrestare una sessione dopo averla avviata. Ad esempio, se noti un problema mentre i tuoi test sono in esecuzione è possibile riavviare la sessione con uno script di prova aggiornato.

Puoi utilizzare la console o l'API Device Farm per interrompere un'esecuzione. AWS CLI

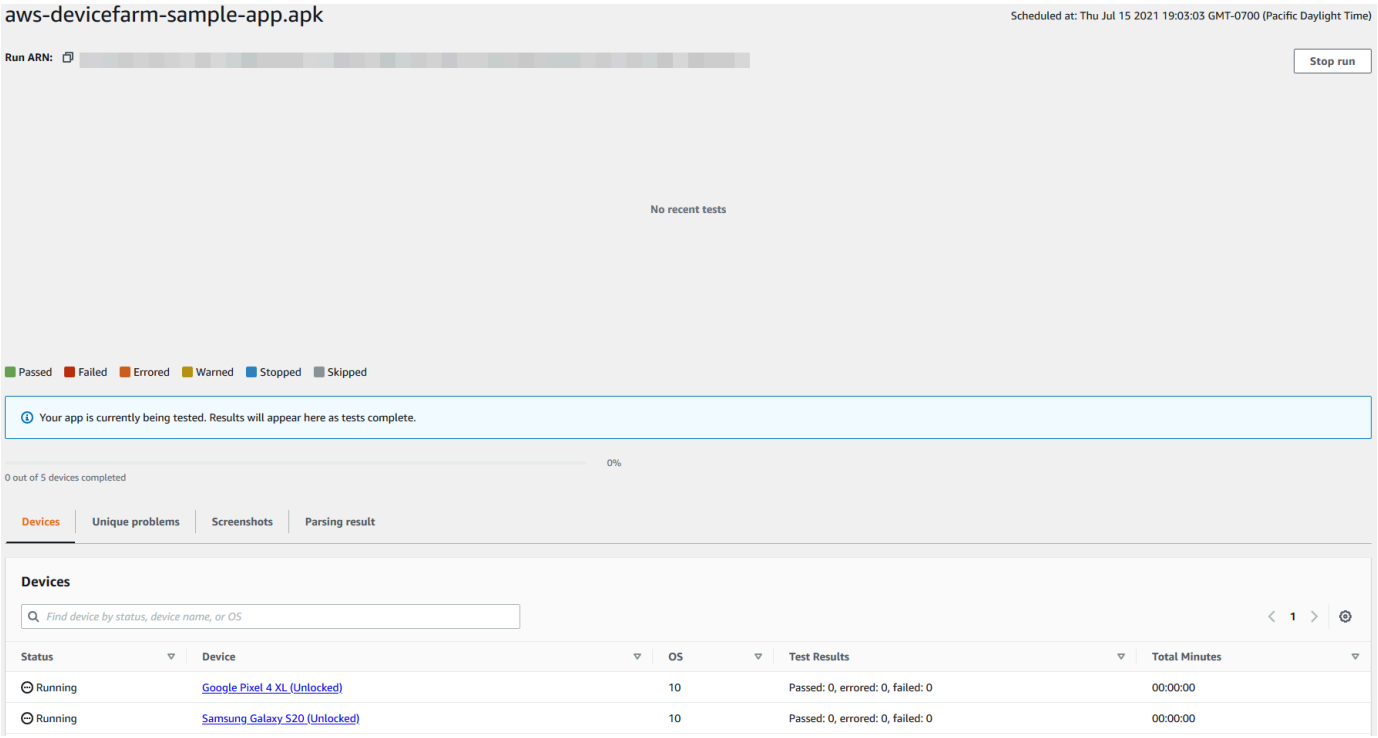
Argomenti

- Interrompi una corsa (console)
- Interrompi una corsa ()AWS CLI
- Interrompere un'esecuzione (API)

Interrompi una corsa (console)

1. Accedere alla console Device Farm all'indirizzo https://console.aws.amazon.com/devicefarm.
2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.
3. Scegliete il progetto in cui avete un test attivo.
4. Nella pagina Test automatici, scegli l'esecuzione del test.

   L'icona in sospeso o in esecuzione dovrebbe apparire a sinistra del nome del dispositivo.

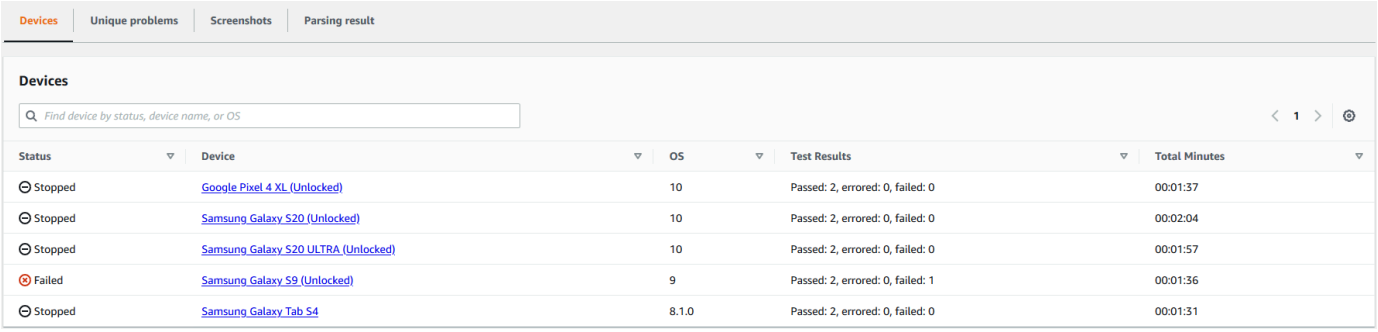


5. Selezionare Stop run (Arresta sessione).

Dopo poco tempo, accanto al nome del dispositivo viene visualizzata un'icona con un cerchio rosso con un segno meno all'interno. Quando la corsa viene interrotta, il colore dell'icona cambia da rosso a nero.

> **Important**
>
> Se un test è già stato eseguito, Device Farm non può fermarlo. Se è in corso un test, Device Farm lo interrompe. Il totale dei minuti che saranno fatturati appare nella sezione Devices (Dispositivi). Inoltre, ti verranno fatturati anche i minuti totali che Device Farm impiega per eseguire la suite di installazione e la suite di smontaggio. Per ulteriori informazioni, consulta Prezzi di Device Farm.

L'immagine seguente mostra un esempio della sezione Dispositivi dopo che una sessione di test è stata correttamente arrestata.



Interrompi una corsa ()AWS CLI

Puoi eseguire il comando seguente per interrompere l'esecuzione del test specificata, dove si *myARN* trova l'Amazon Resource Name (ARN) dell'esecuzione del test.

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

Verrà visualizzato un output simile al seguente:

```
{
    "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
        }
    }
}
```

Per ottenere l'ARN della tua sessione, esegui il comando `list-runs`. L'output visualizzato dovrebbe essere simile al seguente:

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

Per informazioni sull'utilizzo di Device Farm con AWS CLI, vedereAWS CLI riferimento.

Interrompere un'esecuzione (API)

- Richiama l'StopRunoperazione per l'esecuzione del test.

Per informazioni sull'utilizzo dell'API Device Farm, vedereAutomazione di Device Farm.

Visualizzazione di un elenco di esecuzioni in AWS Device Farm

È possibile utilizzare la console o l'API Device Farm per visualizzare un elenco di esecuzioni per un progetto. AWS CLI

Argomenti

- Visualizza un elenco di esecuzioni (console)
- Visualizza un elenco di esecuzioni (AWS CLI)
- Visualizzare un elenco di esecuzioni (API)

Visualizza un elenco di esecuzioni (console)

1. Accedere alla console Device Farm all'indirizzo https://console.aws.amazon.com/devicefarm.
2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.
3. Nell'elenco dei progetti, seleziona il progetto che corrisponde all'elenco che desideri visualizzare.

> **Tip**
>
> Puoi utilizzare la barra di ricerca per filtrare l'elenco dei progetti per nome.

Visualizza un elenco di esecuzioni (AWS CLI)

- Esegui il comando list-runs.

  Per visualizzare informazioni su una singola esecuzione, esegui il comando get-run.

Per informazioni sull'utilizzo di Device Farm con AWS CLI, vedereAWS CLI riferimento.

Visualizzare un elenco di esecuzioni (API)

- Chiamata dell'API `ListRuns`.

  Per visualizzare informazioni su una singola esecuzione, chiama l'API `GetRun`.

Per informazioni sull'API Device Farm, vedereAutomazione di Device Farm.

Creazione di un pool di dispositivi in AWS Device Farm

È possibile utilizzare la console o l'API Device Farm per creare un pool di dispositivi. AWS CLI

Argomenti

Prerequisiti

- Crea una corsa nella console Device Farm. Segui le istruzioni in Creazione di un'esecuzione di test in Device Farm. Una volta alla pagina Select devices (Seleziona dispositivi), continuare con le istruzioni in questa sezione.

Crea un pool di dispositivi (console)

1. Nella pagina Progetti, scegli il tuo progetto. Nella pagina dei dettagli del progetto, scegli Impostazioni del progetto. Nella scheda Device pool, scegli Crea pool di dispositivi.
2. Per Name (Nome), immettere un nome che renda semplice identificare il pool di dispositivi.
3. Per Description (Descrizione), immettere una descrizione che renda semplice identificare il pool di dispositivi.
4. Se si desidera utilizzare uno o più criteri di selezione per i dispositivi in questo pool di dispositivi, eseguire quando segue:

   a. Scegli Crea pool dinamico di dispositivi.

b. Scegli Aggiungi una regola.

c. Per Campo (primo elenco a discesa), scegli una delle seguenti opzioni:

- Per includere i dispositivi in base al nome del produttore, scegli Produttore del dispositivo.
- Per includere i dispositivi in base al fattore di forma (tablet o telefono), scegli Fattore di forma.
- Per includere i dispositivi in base al loro stato di disponibilità in base al carico, scegli Disponibilità.
- Per includere solo dispositivi pubblici o privati, scegli Fleet Type.
- Per includere i dispositivi in base al loro sistema operativo, scegli Piattaforma.
- Alcuni dispositivi hanno un'etichetta o una descrizione aggiuntiva sul dispositivo. Puoi trovare i dispositivi in base al contenuto delle etichette selezionando le etichette delle istanze.
- Per includere i dispositivi in base alla versione del sistema operativo, scegli Versione del sistema operativo.
- Per includere i dispositivi in base al modello, scegli Modello.

d. Per Operatore (secondo elenco a discesa), scegliete un'operazione logica (EQUALS, CONTAINS, ecc.) per includere i dispositivi in base alla query. Ad esempio, puoi scegliere di includere *`Availability EQUALS AVAILABLE`* i dispositivi che attualmente hanno lo stato. `Available`

e. In Valore (terzo elenco a discesa), inserisci o scegli il valore che desideri specificare per i valori Campo e Operatore. I valori sono limitati in base alla scelta del campo. Ad esempio, se scegli Platform for Field, le uniche selezioni disponibili sono ANDROID e IOS. Allo stesso modo, se scegli Form Factor for Field, le uniche selezioni disponibili sono TELEFONO e TABLET.

f. Per aggiungere un'altra regola, scegli Aggiungi una regola.

Dopo aver creato la prima regola, nell'elenco dei dispositivi, la casella accanto a ciascun dispositivo che corrisponde alla regola viene selezionata. Dopo aver creato o modificato le regole, nell'elenco dei dispositivi, la casella accanto a ciascun dispositivo che corrisponde a quelle regole combinate viene selezionata. I dispositivi con caselle selezionate sono inclusi nel pool di dispositivi. I dispositivi con caselle non selezionate vengono esclusi.

g. In Numero massimo di dispositivi, inserisci il numero di dispositivi che desideri utilizzare nel tuo pool di dispositivi. Se non inserisci il numero massimo di dispositivi, Device Farm sceglierà tutti i dispositivi del parco dispositivi che corrispondono alle regole che hai creato.

Per evitare costi aggiuntivi, imposta questo numero su un importo che corrisponda ai tuoi requisiti effettivi di esecuzione parallela e varietà di dispositivi.

h. Per eliminare una regola, scegli Rimuovi regola.

5. Se desideri includere o escludere manualmente singoli dispositivi, procedi come segue:

   a. Scegli Crea pool di dispositivi statici.

   b. Seleziona o deseleziona la casella accanto a ciascun dispositivo. È possibile selezionare o deselezionare le caselle solo se non si dispone di tutte le regole specificate.

6. Se si desidera includere o escludere tutti i dispositivi visualizzati, selezionare o deselezionare la casella nella riga di intestazione della colonna dell'elenco. Se desideri visualizzare solo le istanze private del dispositivo, scegli Visualizza solo le istanze private del dispositivo.

> **Important**
>
> Anche se è possibile utilizzare le caselle nella riga dell'intestazione della colonna per modificare l'elenco dei dispositivi visualizzati, non significa che i dispositivi visualizzati rimanenti siano gli unici inclusi o esclusi. Per confermare quali dispositivi sono inclusi o esclusi, assicurarsi di deselezionare i contenuti di tutte le caselle nella riga di intestazione della colonna, quindi esplorare le caselle.

7. Scegli Create (Crea).

Crea un pool di dispositivi ()AWS CLI

> **Tip**
>
> Se non inserisci il numero massimo di dispositivi, Device Farm sceglierà tutti i dispositivi del parco dispositivi che corrispondono alle regole che hai creato. Per evitare costi aggiuntivi, imposta questo numero su un importo che corrisponda ai tuoi requisiti effettivi di esecuzione parallela e varietà di dispositivi.

- Esegui il comando create-device-pool.

Per informazioni sull'utilizzo di Device Farm con AWS CLI, vedereAWS CLI riferimento.

Creare un pool di dispositivi (API)

> **Tip**
>
> Se non inserisci il numero massimo di dispositivi, Device Farm sceglierà tutti i dispositivi del parco dispositivi che corrispondono alle regole che hai creato. Per evitare costi aggiuntivi, imposta questo numero su un importo che corrisponda ai tuoi requisiti effettivi di esecuzione parallela e varietà di dispositivi.

- Chiamata dell'API `CreateDevicePool`.

Per informazioni sull'utilizzo dell'API Device Farm, vedereAutomazione di Device Farm.

Analisi dei risultati dei test in AWS Device Farm

Nell'ambiente di test standard, è possibile utilizzare la console Device Farm per visualizzare i report di ogni test durante l'esecuzione del test. La visualizzazione dei report ti aiuta a capire quali test sono stati superati o meno e ti fornisce dettagli sulle prestazioni e sul comportamento dell'app in diverse configurazioni di dispositivi.

Device Farm raccoglie anche altri elementi come file, log e immagini che è possibile scaricare al termine del test. Queste informazioni possono aiutarti ad analizzare il comportamento dell'app su dispositivi reali, a identificare problemi o bug e a diagnosticare i problemi.

Argomenti

- Visualizzazione dei report dei test in Device Farm
- Scaricamento di artefatti in Device Farm

Visualizzazione dei report dei test in Device Farm

Usa la console Device Farm per visualizzare i report dei test. Per ulteriori informazioni, consulta Report in AWS Device Farm.

Argomenti

- Prerequisiti
- Visualizza i report

- Stati dei risultati del test Device Farm

Prerequisiti

Configura un'esecuzione di un test e verifica che sia completa.

1. Per creare un'esecuzione, consulta Creazione di un'esecuzione di test in Device Farm e torna a questa pagina.
2. Verifica che l'esecuzione sia completa. Durante l'esecuzione del test, la console Device Farm visualizza un'icona in sospeso  per le esecuzioni in corso. Ogni dispositivo in esecuzione inizierà anche con l'icona in sospeso, quindi passerà all'icon  in esecuzione all'inizio del test. Al termine di ogni test, accanto al nome del dispositivo viene visualizzata l'icona del risultato del test. Una volta completati tutti i test, l'icona in sospeso accanto all'esecuzione diventa l'icona del risultato del test. Per ulteriori informazioni, consulta Stati dei risultati del test Device Farm.

Visualizza i report

È possibile visualizzare i risultati del test nella console Device Farm.

Argomenti

- Visualizza la pagina di riepilogo dell'esecuzione del test
- Visualizza segnalazioni di problemi uniche
- Visualizza i report sui dispositivi
- Visualizza i report della suite di test
- Visualizzazione dei report di test
- Visualizza le informazioni di registro relative a un problema, un dispositivo, una suite o un test in un rapporto

Visualizza la pagina di riepilogo dell'esecuzione del test

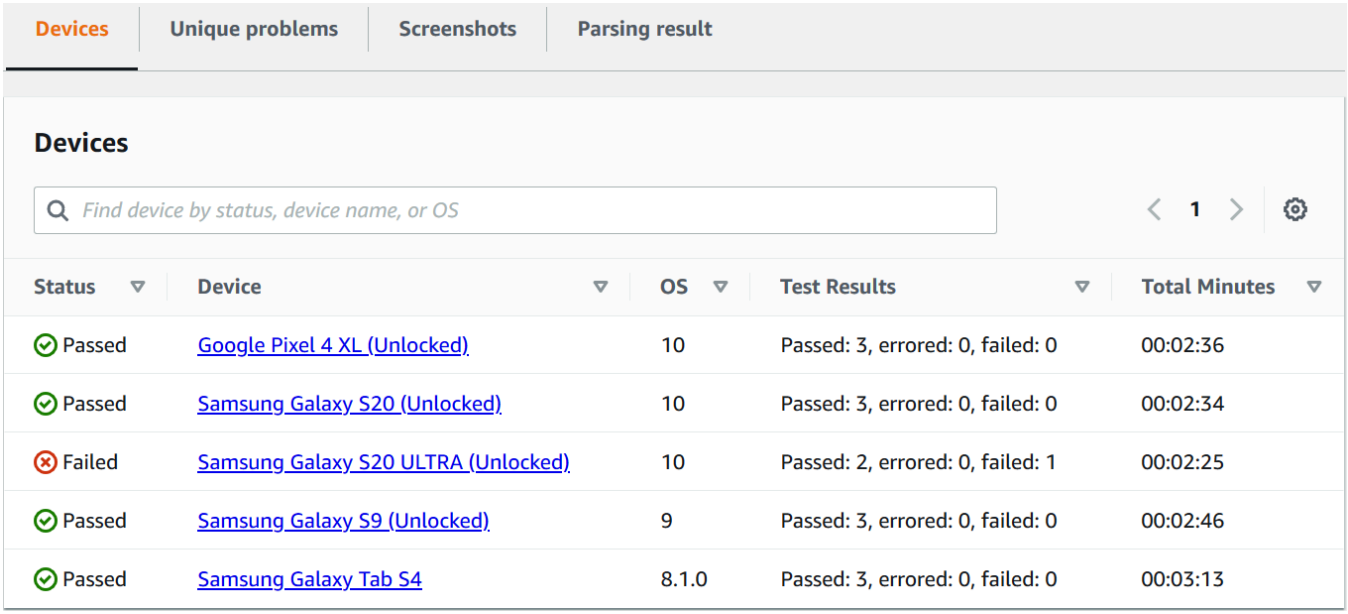
1. Accedere alla console Device Farm all'indirizzo https://console.aws.amazon.com/devicefarm.

2. Nel riquadro di navigazione, scegli Mobile Device Testing, quindi scegli Progetti.
3. Nell'elenco di progetti, scegli il progetto da eseguire.

> **Tip**
>
> Per filtrare l'elenco dei progetti per nome, usa la barra di ricerca.

4. Scegli un'esecuzione completata per visualizzarne la pagina con il report dei riepiloghi.
5. La pagina di riepilogo delle esecuzioni dei test mostra una panoramica dei risultati dei test.

   - La sezione Unique problems (Problemi univoci) elenca gli avvisi e gli errori univoci. Per visualizzare i problemi univoci, segui le istruzioni contenute in Visualizza segnalazioni di problemi uniche.
   - La sezione Devices (Dispositivi) mostra il numero totale di test, per risultato, per ogni dispositivo.



   In questo esempio, ci sono diversi dispositivi. Nella prima voce della tabella, il dispositivo Google Pixel 4 XL con Android versione 10 riporta tre test riusciti, la cui esecuzione ha richiesto 02:36 minuti.

   Per visualizzare i risultati per dispositivo, segui le istruzioni contenute in Visualizza i report sui dispositivi.

- La sezione Screenshots mostra un elenco di tutte le schermate acquisite da Device Farm durante l'esecuzione, raggruppate per dispositivo.
- Nella sezione Risultati dell'analisi, puoi scaricare il risultato dell'analisi.

Visualizza segnalazioni di problemi uniche

1. In Unique problems (Problemi univoci), scegli il problema che desideri visualizzare.
2. Scegli il dispositivo. Il report include informazioni sul problema.

   La sezione Video mostra una registrazione video scaricabile del test.

   La sezione Risultato mostra il risultato del test. Lo stato è rappresentato dall'icona del risultato. Per ulteriori informazioni, consulta Stati di un singolo test.

   La sezione Logs mostra tutte le informazioni registrate da Device Farm durante il test. Per visualizzare queste informazioni, segui le istruzioni contenute in Visualizza le informazioni di registro relative a un problema, un dispositivo, una suite o un test in un rapporto.

   La scheda File mostra un elenco di tutti i file associati al test (come i file di registro) che è possibile scaricare. Per scaricare un file, scegli il relativo link nell'elenco.

   La scheda Screenshots mostra un elenco di tutte le schermate acquisite da Device Farm durante il test.

Visualizza i report sui dispositivi

- Nella sezione Devices (Dispositivi), scegli il dispositivo.

  La sezione Video mostra una registrazione video scaricabile del test.

  La sezione Suite visualizza una tabella contenente informazioni sulle suite per il dispositivo.

  In questa tabella, la colonna Risultati dei test riepiloga il numero di test per risultato per ciascuna delle suite di test eseguite sul dispositivo. Questi dati hanno anche una componente grafica. Per ulteriori informazioni, consulta Stati per test multipli.

  Per visualizzare i risultati completi per suite, segui le istruzioni riportate inVisualizza i report della suite di test.

La sezione Logs mostra tutte le informazioni che Device Farm ha registrato per il dispositivo durante l'esecuzione. Per visualizzare queste informazioni, segui le istruzioni contenute in Visualizza le informazioni di registro relative a un problema, un dispositivo, una suite o un test in un rapporto.

La sezione File visualizza un elenco di suite per il dispositivo e tutti i file associati (come i file di registro) che è possibile scaricare. Per scaricare un file, scegli il relativo link nell'elenco.

La sezione Screenshots mostra un elenco di tutte le schermate acquisite da Device Farm durante l'esecuzione per il dispositivo, raggruppate per suite.

Visualizza i report della suite di test

1. Nella sezione Devices (Dispositivi), scegli il dispositivo.
2. Nella sezione Suite, scegli la suite dalla tabella.

   La sezione Video mostra una registrazione video scaricabile del test.

   La sezione Test mostra una tabella contenente informazioni sui test della suite.

   Nella tabella, la colonna Risultati dei test mostra il risultato. Questi dati hanno anche una componente grafica. Per ulteriori informazioni, consulta Stati per test multipli.

   Per visualizzare i risultati completi dei test, segui le istruzioni riportate inVisualizzazione dei report di test.

   La sezione Logs mostra tutte le informazioni registrate da Device Farm durante l'esecuzione della suite. Per visualizzare queste informazioni, segui le istruzioni contenute in Visualizza le informazioni di registro relative a un problema, un dispositivo, una suite o un test in un rapporto.

   La sezione File mostra un elenco di test per la suite e tutti i file associati (come i file di registro) che è possibile scaricare. Per scaricare un file, scegli il relativo link nell'elenco.

   La sezione Screenshots mostra un elenco di tutte le schermate acquisite da Device Farm durante l'esecuzione della suite, raggruppate per test.

Visualizzazione dei report di test

1. Nella sezione Devices (Dispositivi), scegli il dispositivo.

2. Nella sezione Suites (Suite), scegli la suite.
3. Nella sezione Test, scegli il test.
4. La sezione Video mostra una registrazione video scaricabile del test.

   La sezione Risultato mostra il risultato del test. Lo stato è rappresentato dall'icona del risultato. Per ulteriori informazioni, consulta Stati di un singolo test.

   La sezione Logs mostra tutte le informazioni registrate da Device Farm durante il test. Per visualizzare queste informazioni, segui le istruzioni contenute in Visualizza le informazioni di registro relative a un problema, un dispositivo, una suite o un test in un rapporto.

   La scheda File mostra un elenco di tutti i file associati al test (come i file di registro) che è possibile scaricare. Per scaricare un file, scegli il relativo link nell'elenco.

   La scheda Screenshots mostra un elenco di tutte le schermate acquisite da Device Farm durante il test.

Visualizza le informazioni di registro relative a un problema, un dispositivo, una suite o un test in un rapporto

La sezione Registri mostra le seguenti informazioni:

- Source (Origine) indica l'origine di una voce di log. I valori possibili includono:
  - Harness rappresenta una voce di registro creata da Device Farm. Queste voci di registro vengono in genere create durante eventi di avvio e arresto.
  - Device rappresenta una voce di registro creata dal dispositivo. Per Android, queste voci di registro sono compatibili con logcat. Per iOS, queste voci di registro sono compatibili con syslog.
  - Test indica una voce di registro creata da un test o dal relativo framework.
- Time (Tempo) indica il tempo trascorso tra la prima voce di log e questa. L'ora è espressa in un *MM:SS.SSS* formato, dove *M* rappresenta i minuti e *S* rappresenta i secondi.
- PID indica l'identificatore del processo (PID) che ha creato la voce di log. Tutte le voci di log create da un'app su un dispositivo hanno lo stesso PID.
- Level (Livello) indica il livello registrato per la voce di log. Ad esempio, `Logger.debug("This is a message!")` registra il livello `Debug`. I valori possibili sono:
  - Avviso
  - Critico

- Esegui il debug
- Emergenza
- Errore
- Errore
- Non riuscito
- Informazioni
- Interno
- Comunicazione
- Superato
- Saltato
- Arrestate
- Modalità dettagliata
- Notifica
- Attenzione

- Tag indica metadati arbitrari per la voce di log. Ad esempio, il logcat Android può servirsene per descrivere quale parte del sistema ha creato la voce di log (ad esempio, `ActivityManager`).
- Message (Messaggio) indica il messaggio o i dati per la voce di registro. Ad esempio, `Logger.debug("Hello, World!")` registra il valore Message (Messaggio) `"Hello, World!"`.

Per visualizzare solo una parte delle informazioni:

- Per mostrare tutte le voci di registro che corrispondono a un valore per una colonna specifica, inserisci il valore nella barra di ricerca. Ad esempio, per mostrare tutte le voci di registro con un valore Source pari a`Harness`, **`Harness`** inseriscilo nella barra di ricerca.
- Per rimuovere tutti i caratteri da un riquadro dell'intestazione di una colonna, seleziona la X in quel riquadro in questione. Rimuovere tutti i caratteri da una casella di intestazione di colonna equivale a inserirli ***** in quella casella di intestazione di colonna.

Per scaricare tutte le informazioni di registro per il dispositivo, incluse tutte le suite e i test che hai eseguito, scegli Scarica registri.

Stati dei risultati del test Device Farm

La console Device Farm mostra icone che consentono di valutare rapidamente lo stato dell'esecuzione del test completata. Per ulteriori informazioni sui test in Device Farm, vedereReport in AWS Device Farm.

Argomenti

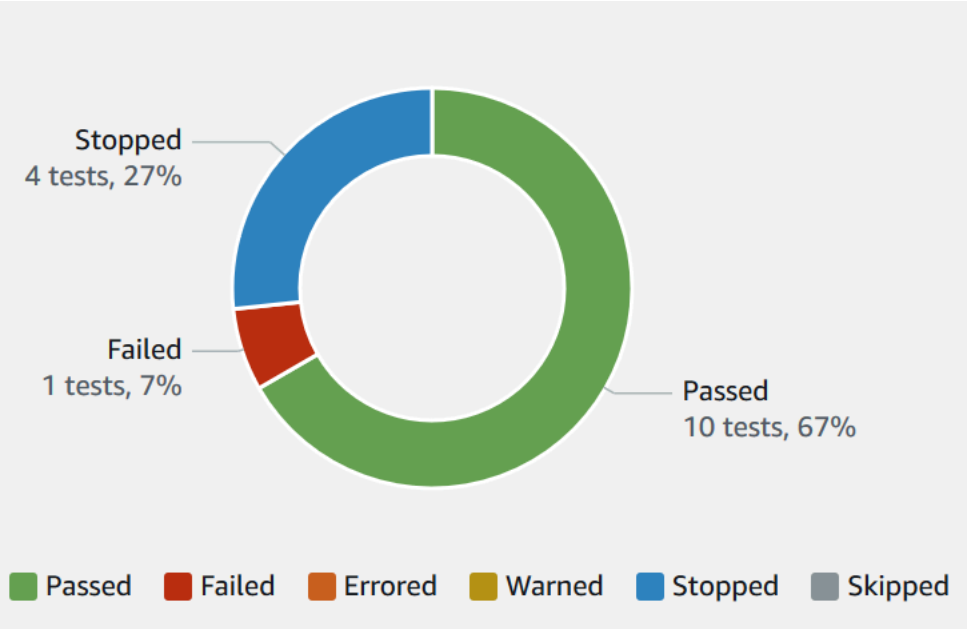
Stati di un singolo test

Per i report che descrivono un singolo test, Device Farm visualizza un'icona che rappresenta lo stato del risultato del test:

Description	Icon
Il test ha avuto esito positivo.	
Il test ha avuto esito negativo.	
Device Farm ha saltato il test.	
Il test si è interrotto.	
Device Farm ha restituito un avviso.	
Device Farm ha restituito un errore.	

Stati per test multipli

Se si sceglie un'esecuzione completata, Device Farm visualizza un grafico riassuntivo che mostra la percentuale di test nei vari stati.



Ad esempio, questo grafico dei risultati dell'esecuzione del test mostra che nella corsa sono stati interrotti 4 test interrotti, 1 test fallito e 10 test riusciti.

I grafici sono sempre codificati a colori ed etichettati.

Scaricamento di artefatti in Device Farm

Device Farm raccoglie artefatti come report, file di registro e immagini per ogni test in esecuzione.

Puoi scaricare artefatti creati durante la sessione di test:

File

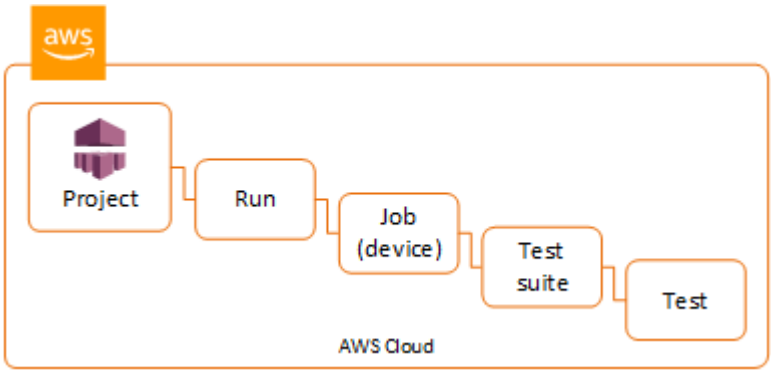
File generati durante l'esecuzione del test, inclusi i report di Device Farm. Per ulteriori informazioni, consulta Visualizzazione dei report dei test in Device Farm.

Log

Output da ciascun test nella sessione.

Screenshot

Immagini di schermata registrate per ogni test nella sessione.



Scarica artefatti (console)

1. Nella pagina del report della sessione di test, da Devices (Dispositivi), selezionare un dispositivo mobile.
2. Per scaricare un file, sceglierne uno da Files (File).
3. Per scaricare i log dalla sessione di test, da Logs (Log), selezionare Download logs (Scarica log).
4. Per scaricare uno screenshot, scegliere uno screenshot da Screenshots (Screenshot).

Per ulteriori informazioni su come scaricare gli artefatti in un ambiente di test personalizzato, consulta Scaricamento di artefatti in un ambiente di test personalizzato.

Scarica gli artefatti ()AWS CLI

Puoi usare il AWS CLI per elencare gli artefatti del test eseguito.

Argomenti

- Fase 1: Ottieni i tuoi Amazon Resource Names (ARN)
- Fase 2: Elenca i tuoi artefatti
- Passaggio 3: scarica i tuoi artefatti

Fase 1: Ottieni i tuoi Amazon Resource Names (ARN)

Puoi elencare i tuoi artefatti per sessione, lavoro, suite di test o test. Ti occorre l'ARN corrispondente. Questa tabella mostra l'ARN di input per ciascuno dei comandi dell' AWS CLI elenco:

AWS CLI Comando di elenco	ARN richiesto
list-projects	Questo comando restituisce tutti i progetti e non richiede un ARN.
list-runs	`project`
list-jobs	`run`
list-suites	`job`
list-tests	`suite`

Ad esempio, per trovare l'ARN di un test, esegui list-tests utilizzando l'ARN della tua suite di test come parametro di input.

Esempio:

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

La risposta include un ARN di test per ogni test nella suite di test.

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

Fase 2: Elenca i tuoi artefatti

Il comando AWS CLI list-artifacts restituisce un elenco di artefatti, come file, schermate e registri. Ogni artefatto dispone di un URL in modo che tu possa scaricare il file.

- Chiama list-artifacts specificando l'ARN di una sessione, un lavoro, una suite di test o un test. Specifica un tipo di FILE, LOG o SCREENSHOT.

  Questo esempio restituisce un URL di download per ogni artefatto disponibile per un singolo test:

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

  La risposta contiene un URL di download per ogni artefatto.

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

Passaggio 3: scarica i tuoi artefatti

- Scarica i tuoi artefatti utilizzando l'URL dalla fase precedente. Questo esempio utilizza curl per scaricare un file di output Android Appium Java:

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

Scarica gli artefatti (API)

Il ListArtifactsmetodo API Device Farm restituisce un elenco di elementi, come file, schermate e registri. Ogni artefatto dispone di un URL in modo che tu possa scaricare il file.

Scaricamento di artefatti in un ambiente di test personalizzato

In un ambiente di test personalizzato, Device Farm raccoglie artefatti come report personalizzati, file di registro e immagini. Questi artefatti sono disponibili per ciascun dispositivo nella sessione di test.

Puoi scaricare questi artefatti creati durante la sessione di test:

Output di specifica di test

L'output derivante dall'esecuzione dei comandi nel file YAML di specifica di test.

Artefatti personalizzati

Un file compresso che contiene gli artefatti della sessione di test. È configurato nella sezione artifact: (artefatti:) sezione del tuo file YAML di specifica di test.

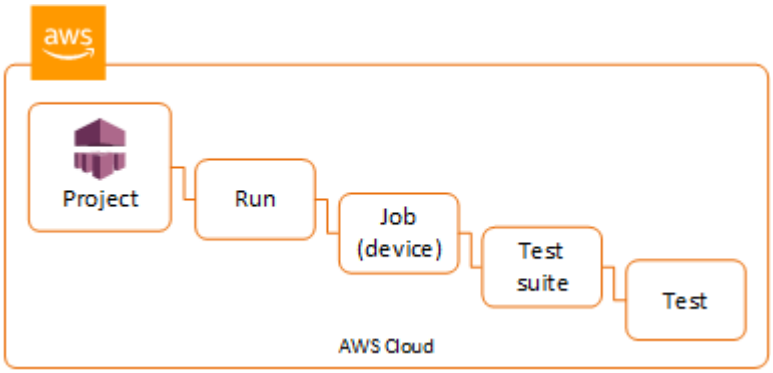
Script della shell di specifica di test

Un file di script della shell intermedio creato dal tuo file YAML. Dato che è utilizzato nella sessione di test, il file di script della shell può essere utilizzato per il debug del file YAML.

File di specifica di test

Il file YAML utilizzato nella sessione di test.

Per ulteriori informazioni, consulta Scaricamento di artefatti in Device Farm.


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

Etichettatura delle risorse di AWS Device Farm

AWS Device Farm funziona con l'API AWS Resource Groups Tagging. Questa API consente di gestire le risorse nel tuo account AWS con tag. È possibile aggiungere tag alle risorse, ad esempio progetti ed esecuzioni di test.

È possibile usare i tag per:

- organizzare le fatture AWS in modo che riflettano la tua struttura dei costi. Per eseguire questa operazione, registrati per far sì che la fattura del tuo account AWS includa i valori di chiave di tag. Per visualizzare il costo delle risorse combinate, puoi organizzare le informazioni di fatturazione in base alle risorse con gli stessi valori di chiave di tag. Puoi ad esempio applicare tag a numerose risorse con un nome di applicazione specifico, quindi organizzare le informazioni di fatturazione per visualizzare il costo totale dell'applicazione in più servizi. Per ulteriori informazioni, consultare l'argomento relativo a tagging e allocazione dei costi nelle informazioni relative a AWS Billing and Cost Management.
- Controlla l'accesso tramite i criteri IAM. A tale scopo, crea una policy che consenta l'accesso a una risorsa o a un set di risorse utilizzando una condizione del valore del tag.
- Identifica e gestisci le esecuzioni con determinate proprietà come tag, ad esempio il ramo utilizzato per il test.

Per ulteriori informazioni sulle risorse di tagging, vedere il white paper Best practice relative al tagging.

Argomenti

Applicazione di tag alle risorse

L'API per l'applicazione di tag a gruppi di risorse AWS consente di aggiungere, rimuovere o modificare i tag sulle risorse. Per ulteriori informazioni, consulta la Documentazione di riferimento delle API di tagging dei gruppi di risorse.

Per applicare tag a una risorsa, utilizzare l'operazione `TagResources` dall'endpoint `resourcegroupstaggingapi`. Questa operazione richiede un elenco dei ARNs servizi supportati e un elenco di coppie chiave-valore. Il valore è facoltativo. Una stringa vuota indica che non dovrebbe esserci alcun valore per quel tag. Ad esempio, il seguente esempio di Python etichetta una serie di progetti ARNs con il tag `build-config` con il valore: `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"})
```

Il valore del tag non è obbligatorio. Per impostare un tag senza valore, utilizzare una stringa vuota ("") quando si specifica un valore. Un tag può avere un solo valore. Qualsiasi valore precedente che un tag ha per una risorsa verrà sovrascritto con il nuovo valore.

Ricerca di risorse per tag

Per eseguire la ricerca delle risorse in base ai tag, utilizzare l'operazione `GetResources` dall'endpoint `resourcegrouptaggingapi`. Questa operazione utilizza una serie di filtri, nessuno dei quali è necessario, e restituisce le risorse che corrispondono ai criteri specificati. Senza filtri, vengono restituite tutte le risorse a cui sono applicati tag. L'operazione `GetResources` consente di filtrare le risorse in base a

- Valore di tag
- Tipo di risorsa (ad esempio, `devicefarm:run`)

Per ulteriori informazioni, consulta la Documentazione di riferimento delle API di tagging dei gruppi di risorse.

L'esempio seguente cerca le sessioni (`devicefarm:testgrid-session`risorse) di test del browser desktop Device Farm con il tag `stack` che ha il valore`production`:

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

Rimozione dei tag dalle risorse

Per rimuovere un tag, utilizzare l'operazione `UntagResources`, specificando un elenco di risorse e i tag da rimuovere:

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

Framework di test e test integrati in AWS Device Farm

Questa sezione descrive il supporto di Device Farm per i framework di test e i tipi di test integrati.

Device Farm esegue test automatici caricando l'app e i test in un bucket Amazon S3 sicuro gestito dal servizio. Una volta caricato, attiva l'infrastruttura sottostante, inclusi gli host di test gestiti dai servizi, ed esegue i test in parallelo su più dispositivi. I risultati dei test vengono archiviati in un bucket S3 gestito dal servizio. Questa architettura si chiama esecuzione lato servizio ed è un modo rapido ed efficiente per eseguire test su host fisicamente vicini al dispositivo, senza dover gestire personalmente l'infrastruttura dell'host di test. Questo approccio si adatta bene ai test su molti dispositivi in modo indipendente, nonché ai test nel contesto di una pipeline. CI/CD

Per ulteriori informazioni su come Device Farm esegue i test, vedereAmbienti di test in AWS Device Farm.

> **Note**
>
> Per i tester Appium, potresti preferire eseguire i test Appium dal tuo ambiente locale. Con una sessione di accesso remoto, puoi eseguire test Appium lato client. Per ulteriori informazioni, consulta Appium testing lato client.

Framework di test

Device Farm supporta questi framework di test per l'automazione mobile:

Framework di test delle applicazioni Android

- Test Appium automatici
- Instrumentation

Framework di test delle applicazioni iOS

- Test Appium automatici
- XCTest
- XCTest INTERFACCIA UTENTE

Framework di test delle applicazioni Web

Le applicazioni Web sono supportate utilizzando Appium. Per ulteriori informazioni su come trasferire i test su Appium, consulta Esegui automaticamente i test Appium in Device Farm.

Framework in un ambiente di test personalizzato

Device Farm non fornisce supporto per la personalizzazione dell'ambiente di test per il XCTest framework. Per ulteriori informazioni, consulta Ambienti di test personalizzati in AWS Device Farm.

Supporto per la versione Appium

Per i test eseguiti in un ambiente personalizzato, Device Farm supporta la versione 1 di Appium. Per ulteriori informazioni, consulta Ambienti di test in AWS Device Farm.

Tipi di test integrati

Con i test integrati, puoi testare la tua applicazione su più dispositivi senza dover scrivere e gestire script di automazione dei test. Device Farm offre un tipo di test integrato:

- Integrato: fuzz (Android e iOS)

Esegui automaticamente i test Appium in Device Farm

> **Note**
>
> Questa pagina descrive l'esecuzione dei test Appium nell'ambiente di esecuzione lato server gestito di Device Farm. Per eseguire i test Appium dal tuo ambiente locale lato client durante una sessione di accesso remoto, consulta Appium testing lato client.

Questa sezione descrive come configurare, impacchettare e caricare i test Appium per l'esecuzione nell'ambiente lato server gestito di Device Farm. Appium è uno strumento open source per automatizzare le applicazioni web native e mobili. Per ulteriori informazioni, vedere Introduzione ad Appium sul sito Web di Appium.

Per un'app di esempio e i collegamenti ai test di lavoro, consulta Device Farm Sample App per Android e Device Farm Sample App per iOS su GitHub.

Per ulteriori informazioni sui test in Device Farm e sul funzionamento lato server, vedere. Framework di test e test integrati in AWS Device Farm

Selezione di una versione di Appium

> **Note**
>
> Il supporto per versioni Appium specifiche, driver Appium o programmazione SDKs dipenderà dal dispositivo e dall'host di test selezionati per l'esecuzione del test.

Gli host di test Device Farm sono preinstallati con Appium per consentire una configurazione più rapida dei test per casi d'uso più semplici. Tuttavia, l'uso del file test spec consente di installare diverse versioni di Appium, se necessario.

Scenario 1: versione Appium preconfigurata

Device Farm è preconfigurato con diverse versioni del server Appium in base all'host di test. L'host è dotato di strumenti che abilitano la versione preconfigurata con il driver predefinito della piattaforma del dispositivo (UiAutomator2 per Android e per XCUITest iOS).

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

Per visualizzare un elenco dei software supportati, consulta l'argomento su. Software supportato in ambienti di test personalizzati

Scenario 2: versione Appium personalizzata

Per selezionare una versione personalizzata di Appium, usa il `npm` comando per installarla. L'esempio seguente mostra come installare l'ultima versione di Appium 2.

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

Scenario 3: Appium su host iOS precedenti

SuHost di test iOS legacy, puoi scegliere versioni specifiche di Appium con. `avm` Ad esempio, per utilizzare il `avm` comando su cui impostare la versione del server Appium`2.1.2`, aggiungi questi comandi al file YAML della specifica di test.

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

Selezione di una WebDriverAgent versione per i test iOS

Per eseguire i test Appium su dispositivi iOS, WebDriverAgent è necessario l'uso di. Questa applicazione deve essere firmata per poter essere installata su dispositivi iOS. Device Farm fornisce versioni prefirmate WebDriverAgent che sono disponibili durante le esecuzioni di ambienti di test personalizzati.

Il seguente frammento di codice può essere utilizzato per selezionare una WebDriverAgent versione in Device Farm all'interno del file delle specifiche di test compatibile con la versione del driver XCTest UI.

```
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;
```

Per ulteriori informazioni su WebDriverAgent, consulta la documentazione di Appium.

Integrazione dei test Appium con Device Farm

Utilizza le seguenti istruzioni per integrare i test Appium con AWS Device Farm. Per ulteriori informazioni sull'utilizzo dei test Appium in Device Farm, vedere. Esegui automaticamente i test Appium in Device Farm

Configura il tuo pacchetto di test Appium

Utilizza le seguenti istruzioni per configurare il pacchetto del test.

Java (JUnit)

1. Modifica `pom.xml` per impostare la confezione su un file JAR:

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

2. Modifica `pom.xml` per utilizzarlo `maven-jar-plugin` per creare i test in un file JAR.

   Il seguente plugin crea il codice sorgente del test (qualsiasi cosa nella `src/test` directory) in un file JAR:

   ```
   <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. Modifica `pom.xml` per utilizzarlo `maven-dependency-plugin` per creare dipendenze come file JAR.

   Il seguente plugin copia le tue dipendenze nella `dependency-jars` directory:

```
<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. Salvate il seguente assembly XML in. `src/main/assembly/zip.xml`

   Il seguente codice XML è una definizione di assembly che, una volta configurata, indica a Maven di creare un file.zip che contenga tutto ciò che si trova nella radice della directory di output della build e della directory: `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. Modifica `pom.xml` per utilizzarlo per `maven-assembly-plugin` impacchettare i test e tutte le dipendenze in un unico file.zip.

   Il seguente plugin utilizza l'assembly precedente per creare un file.zip denominato `zip-with-dependencies` nella directory di output della build ogni volta che viene eseguito: mvn package

```
<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**
>
> Se ricevi un messaggio di errore indicante che l'annotazione non è supportata nella versione 1.3, aggiungi quanto segue a `pom.xml`:
>
> ```
> <plugin>
>   <artifactId>maven-compiler-plugin</artifactId>
>   <configuration>
>     <source>1.7</source>
>     <target>1.7</target>
>   </configuration>
> </plugin>
> ```

Java (TestNG)

1. Modifica `pom.xml` per impostare la confezione su un file JAR:

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

2. Modifica `pom.xml` per utilizzarlo `maven-jar-plugin` per creare i test in un file JAR.

   Il seguente plugin crea il codice sorgente del test (qualsiasi cosa nella `src/test` directory) in un file JAR:

   ```
   <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. Modifica `pom.xml` per utilizzarlo `maven-dependency-plugin` per creare dipendenze come file JAR.

   Il seguente plugin copia le tue dipendenze nella `dependency-jars` directory:

```
<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. Salvate il seguente assembly XML in. `src/main/assembly/zip.xml`

   Il seguente codice XML è una definizione di assembly che, una volta configurata, indica a Maven di creare un file.zip che contenga tutto ciò che si trova nella radice della directory di output della build e della directory: `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. Modifica `pom.xml` per utilizzarlo per `maven-assembly-plugin` impacchettare i test e tutte le dipendenze in un unico file.zip.

   Il seguente plugin utilizza l'assembly precedente per creare un file.zip denominato `zip-with-dependencies` nella directory di output della build ogni volta che viene eseguito: mvn package

```
<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**
>
> Se ricevi un messaggio di errore indicante che l'annotazione non è supportata nella versione 1.3, aggiungi quanto segue a `pom.xml`:
>
> ```
> <plugin>
>   <artifactId>maven-compiler-plugin</artifactId>
>   <configuration>
>     <source>1.7</source>
>     <target>1.7</target>
>   </configuration>
> </plugin>
> ```

Node.JS

Per impacchettare i test di Appium Node.js e caricarli su Device Farm, è necessario installare quanto segue sul computer locale:

- Node Version Manager (nvm)

  Utilizzare questo strumento per sviluppare e creare pacchetti di test in modo che inutili dipendenze non siano incluse nel pacchetto di test.
- Node.js
- npm-bundle (installato a livello globale)

1. Verificare che nvm sia presente

   ```
   command -v nvm
   ```

   Dovresti vedere `nvm` come output.

   Per ulteriori informazioni, vedere nvm on. GitHub
2. Eseguire questo comando per installare Node.js:

   ```
   nvm install node
   ```

   È possibile specificare una determinata versione di Node.js:

```
nvm install 11.4.0
```

3. Verificare che la versione corretta di Node sia in uso:

```
node -v
```

4. Installare npm-bundle globalmente:

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

Python

1. Si consiglia vivamente di configurare un virtualenv Python per lo sviluppo e di impacchettare i test in modo che le dipendenze non necessarie siano escluse dal pacchetto dell'applicazione.

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

> **Tip**
>
> - Non creare un virtualenv Python con l'opzione `--system-site-packages`, in quanto eredita pacchetti dalla directory site-packages globale. Questo può causare l'inclusione di dipendenze nell'ambiente virtuale non richieste dai test.
> - Verificare inoltre che i test non utilizzino dipendenze dipendenti da librerie native, in quanto queste librerie native potrebbero non essere presenti sull'istanza in cui sono eseguiti i test.

2. Installare py.test nell'ambiente virtuale.

```
$ pip install pytest
```

3. Installare il client Appium Python nell'ambiente virtuale.

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

4. A meno che non si specifichi un percorso diverso in modalità personalizzata, Device Farm prevede che i test vengano archiviati in`tests/`. È possibile utilizzare `find` per mostrare tutti i file all'interno di una cartella:

   ```
   $ find tests/
   ```

   Verifica che questi file contengano suite di test che desideri eseguire su Device Farm

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

5. Eseguire questo comando dalla cartella dell'area di lavoro dell'ambiente virtuale in modo da visualizzare un elenco di test senza eseguirli.

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

   Conferma che l'output mostri i test che desideri eseguire su Device Farm.

6. Pulire tutti i file memorizzati nella cache nella cartella dei test:

   ```
   $ 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. Eseguire il comando seguente nell'area di lavoro, per generare il file requirements.txt:

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

Ruby

Per impacchettare i test di Appium Ruby e caricarli su Device Farm, è necessario installare quanto segue sul computer locale:

- Ruby Version Manager (RVM)

  Utilizzare questo strumento a riga di comando per sviluppare e creare pacchetti di test in modo che inutili dipendenze non siano incluse nel pacchetto di test.

- Ruby
- Bundler (viene in genere installato con Ruby).

1. Installare le chiavi richieste, RVM e Ruby. Per istruzioni, consulta Installazione di RVM sul sito Web relativo.

   Una volta completata l'installazione, ricaricare il terminale uscendo e accedendo nuovamente.

   > **Note**
   >
   > RVM è caricato come funzione solo per shell bash.

2. Verifica che rvm sia installato correttamente.

   ```
   command -v rvm
   ```

   Dovresti vedere `rvm` come output.

3. Se vuoi installare una versione specifica di Ruby, ad esempio*2.5.3*, esegui il seguente comando:

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

   Verificare di utilizzare la versione richiesta di Ruby:

   ```
   ruby -v
   ```

4. Configura il bundler per compilare i pacchetti per le piattaforme di test desiderate:

   ```
   bundle config specific_platform true
   ```

5. Aggiorna il file.lock per aggiungere le piattaforme necessarie per eseguire i test.

   - Se stai compilando test da eseguire su dispositivi Android, esegui questo comando per configurare Gemfile in modo che utilizzi le dipendenze per l'host di test Android:

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

- Se stai compilando test da eseguire su dispositivi iOS, esegui questo comando per configurare Gemfile in modo che utilizzi le dipendenze per l'host di test iOS:

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

6. La gemma bundler viene solitamente installata per impostazione predefinita. Se non lo è, installa tale elemento:

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

Crea un file di pacchetto di test compresso

> **Warning**
>
> In Device Farm, la struttura delle cartelle dei file nel pacchetto di test compresso è importante e alcuni strumenti di archiviazione modificheranno implicitamente la struttura del file ZIP. Ti consigliamo di seguire le utilità della riga di comando specificate di seguito anziché utilizzare le utilità di archiviazione integrate nel file manager del desktop locale (come Finder o Windows Explorer).

Ora, raggruppare i test per Device Farm.

Java (JUnit)

Creare i test:

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

Come risultato, verrà creato il file `zip-with-dependencies.zip`. Questo è il pacchetto di test.

Java (TestNG)

Creare i test:

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

Come risultato, verrà creato il file `zip-with-dependencies.zip`. Questo è il pacchetto di test.

Node.JS

1. Verifica il progetto.

   Assicurati di essere nella directory principale del progetto. È possibile visualizzare `package.json` nella directory principale.

2. Esegui il comando per l'installazione delle dipendenze locali.

   ```
   npm install
   ```

   Questo comando crea inoltre una cartella `node_modules` all'interno della directory corrente.

   > **Note**
   >
   > A questo punto dovresti essere in grado di eseguire i test in locale.

3. Esegui il comando per creare pacchetti di file nella cartella corrente .tgz*. Il file è denominato utilizzando la proprietà `name` nel file `package.json`.

   ```
   npm-bundle
   ```

   Questo file tarball (.tgz) contiene tutti i codici e le dipendenze.

4. Esegui il comando per creare un bundle di tarball (* file.tgz) generato nel passo precedente in un singolo archivio compresso:

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

   Questo è il `MyTests.zip` file che si carica su Device Farm con la procedura seguente.

Python

Python 2

Generare un archivio dei pacchetti Python richiesti (chiamato "wheelhouse") usando pip:

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

Creare il pacchetto dei requisiti di wheelhouse, test e pip in un archivio zip per Device Farm:

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

Python 3

Creare il pacchetto dei requisiti di test e pip in un file zip:

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

Ruby

1. Esegui questo comando per creare un ambiente Ruby virtuale:

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

2. Esegui questo comando per utilizzare l'ambiente appena creato:

```
rvm gemset use myGemset
```

3. Consulta il codice sorgente.

   Assicurati di essere nella directory principale del progetto. È possibile visualizzare `Gemfile` nella directory principale.

4. Esegui questo comando per l'installazione delle dipendenze locali e di tutte le gemme da `Gemfile`.

```
bundle install
```

> **Note**
>
> A questo punto dovresti essere in grado di eseguire i test in locale. Utilizzare questo comando per eseguire un test in locale:
>
> ```
> bundle exec $test_command
> ```

5. Crea un pacchetto delle gemme nella cartella `vendor/cache`.

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

6. Eseguire il comando seguente per il bundle del codice sorgente, insieme a tutte le tue dipendenze, in un singolo archivio compresso:

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

Questo è il `MyTests.zip` file che si carica su Device Farm con la procedura seguente.

Carica il tuo pacchetto di test su Device Farm

Puoi usare la console Device Farm per caricare i tuoi test.

1. Accedere alla console Device Farm all'indirizzo https://console.aws.amazon.com/devicefarm.
2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.
3. Se sei un nuovo utente, scegli Nuovo progetto, inserisci un nome per il progetto, quindi scegli Invia.

   Se hai già un progetto, puoi sceglierlo per caricarvi i test.
4. Apri il progetto, quindi scegli Crea esegui.
5. In Impostazioni di esecuzione, assegna al test un nome appropriato. Tale nome può contenere qualsiasi combinazione di spazi o punteggiatura.
6. Per i test nativi di Android e iOS

   In Impostazioni Esegui, scegli app Android se stai testando un'applicazione Android (.apk) o scegli app iOS se stai testando un'applicazione iOS (.ipa). Quindi, in Seleziona app, seleziona Carica la tua app per caricare il pacchetto distribuibile dell'applicazione.

> **Note**
>
> Il file deve essere un file `.apk` Android oppure un file `.ipa` iOS. Le applicazioni iOS devono essere costruite per dispositivi reali, non per il simulatore.

Per i test delle applicazioni Web per dispositivi mobili

In Impostazioni di esecuzione, scegli App Web.

7. In Configura test, nella sezione Seleziona il framework di test, scegli il framework Appium con cui esegui il test, quindi carica il tuo pacchetto di test.
8. Individua e seleziona il file .zip che contiene i test. Il file .zip deve presentare il formato descritto in Configura il tuo pacchetto di test Appium.
9. Segui le istruzioni per selezionare i dispositivi e iniziare l'esecuzione. Per ulteriori informazioni, consulta Creazione di un'esecuzione di test in Device Farm.

> **Note**
>
> Device Farm non modifica i test Appium.

Acquisisci schermate dei tuoi test (opzionale)

Durante i test, è possibile acquisire screenshot.

Device Farm imposta la proprietà `DEVICEFARM_SCREENSHOT_PATH` su percorso pienamente qualificato sul sistema locale di file in cui Device Farm si aspetta il salvataggio degli screenshot Appium. La directory specifica del test in cui sono archiviati gli screenshot è definita in fase di esecuzione. Gli screenshot vengono recuperati automaticamente nei report Device Farm. Per visualizzare gli screenshot, nella console Device Farm, selezionare la sezione Screenshots (Screenshot).

Per ulteriori informazioni sull'acquisizione di screenshot nei test Appium, vedere Acquisizione di screenshot nella documentazione dell'API Appium.

Test Android in AWS Device Farm

Device Farm fornisce supporto per diversi tipi di test di automazione per dispositivi Android e due test integrati.

Per ulteriori informazioni sui test in Device Farm, vedereFramework di test e test integrati in AWS Device Farm.

Framework di test delle applicazioni Android

Sono disponibili i seguenti test per i dispositivi Android.

- Test Appium automatici
- Instrumentation

Tipi di test integrati per Android

È disponibile un tipo di test integrato per i dispositivi Android:

- Integrato: fuzz (Android e iOS)

Strumentazione per Android e AWS Device Farm

Device Farm fornisce supporto per Instrumentation (JUnit, Espresso, Robotium o qualsiasi test basato su strumentazione) per Android.

Device Farm fornisce anche un'applicazione Android di esempio e collegamenti a test di lavoro in tre framework di automazione Android, tra cui Instrumentation (Espresso). L'app di esempio Device Farm per Android è disponibile per il download su GitHub.

Per ulteriori informazioni sui test in Device Farm, vedereFramework di test e test integrati in AWS Device Farm.

Argomenti

- Che cos'è la strumentazione?
- Considerazioni per i test di strumentazione Android
- Analisi dei test in modalità standard
- Integrazione della strumentazione Android con Device Farm

Che cos'è la strumentazione?

Instrumentation di Android consente di invocare metodi di callback nel codice di test in modo da poter eseguire tutto il ciclo di vita di un componente passo-passo, come se si stesse effettuando il debug del componente. Per ulteriori informazioni, consulta Test strumentati nella sezione Tipi e posizioni dei test della documentazione di Android Developer Tools.

Considerazioni per i test di strumentazione Android

Quando utilizzi la strumentazione Android, prendi in considerazione i seguenti consigli e note.

Verifica la compatibilità del sistema operativo Android

Consulta la documentazione di Android per assicurarti che Instrumentation sia compatibile con la versione del tuo sistema operativo Android.

Esecuzione dalla riga di comando

Per eseguire i test di Instrumentation dalla riga di comando, segui la documentazione di Android.

System Animations (Animazioni di sistema)

Secondo la documentazione Android per i test di Espresso, si consiglia di disattivare le animazioni di sistema durante i test su dispositivi reali. Device Farm disattiva automaticamente le impostazioni Window Animation Scale, Transition Animation Scale e Animator Duration Scale quando viene eseguito con il test runner di strumentazione JUnitandroid.support.test.runner.Android Runner.

Test Recorders (Registratori di test)

Device Farm supporta framework, come Robotium, che dispongono record-and-playback di strumenti di scripting.

Analisi dei test in modalità standard

Nella modalità standard di esecuzione, Device Farm analizza la suite di test e identifica le classi e i metodi di test univoci che verrà eseguita. Questo viene fatto tramite uno strumento chiamato Dex Test Parser.

Quando viene fornito un file.apk della strumentazione Android come input, il parser restituisce i nomi completi dei metodi dei test che corrispondono JUnit alle convenzioni 3 e 4. JUnit

Per testarlo in un ambiente locale:

1. Scarica il file `dex-test-parser`binario.
2. Esegui il comando seguente per ottenere l'elenco dei metodi di test che verranno eseguiti su Device Farm:

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

Integrazione della strumentazione Android con Device Farm

> **Note**
>
> Utilizza le seguenti istruzioni per integrare i test della strumentazione Android con AWS Device Farm. Per ulteriori informazioni sull'utilizzo dei test di strumentazione in Device Farm, vedere. Strumentazione per Android e AWS Device Farm

Carica i tuoi test di strumentazione Android

Usa la console Device Farm per caricare i test.

1. Accedere alla console Device Farm all'indirizzo https://console.aws.amazon.com/devicefarm.
2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.
3. Nell'elenco dei progetti, scegli il progetto su cui caricare i test.

   > **Tip**
   >
   > Puoi utilizzare la barra di ricerca per filtrare l'elenco dei progetti per nome.
   > Per creare un progetto, segui le istruzioni indicate nella sezione Creazione di un progetto in AWS Device Farm.

4. Seleziona Crea esegui.
5. In Seleziona app, nella sezione Opzioni di selezione dell'app, seleziona Carica la tua app.
6. Individua e seleziona il file dell'app Android. Il file deve essere di tipo .apk.
7. In Configura test, nella sezione Select test framework, scegli Strumentazione, quindi seleziona Scegli file.
8. Individuare e selezionare il file .apk che contiene i test.
9. Completa le istruzioni rimanenti per selezionare i dispositivi e avviare l'esecuzione.

(Facoltativo) Acquisisci schermate nei test della strumentazione Android

Durante i test di Instrumentation per Android, si possono acquisire screenshot.

Per acquisire screenshot, chiamare uno dei seguenti metodi:

- Per Robotium, chiamare il metodo `takeScreenShot` (ad esempio, `solo.takeScreenShot();`).

- Per Spoon, chiamare il metodo `screenshot`, ad esempio:

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

Durante un'esecuzione di test, Device Farm ottiene schermate dalle seguenti posizioni sui dispositivi, se esistono, e quindi le aggiunge ai rapporti di test:

- `/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`*

Test iOS in AWS Device Farm

Device Farm fornisce supporto per diversi tipi di test di automazione per dispositivi iOS e un test integrato.

Per ulteriori informazioni sui test in Device Farm, vedereFramework di test e test integrati in AWS Device Farm.

Framework di test delle applicazioni iOS

Sono disponibili i seguenti test per i dispositivi iOS.

- Test Appium automatici
- XCTest
- XCTest INTERFACCIA UTENTE

Tipi di test integrati per iOS

Attualmente è disponibile un tipo di test integrato per i dispositivi iOS.

- Integrato: fuzz (Android e iOS)

Integrazione di Device Farm con XCTest iOS

Con Device Farm, puoi utilizzare il XCTest framework per testare la tua app su dispositivi reali. Per ulteriori informazioni in merito XCTest, consulta Testing Basics in Testing with Xcode.

Per eseguire un test, create i pacchetti per l'esecuzione del test e caricate questi pacchetti su Device Farm.

Per ulteriori informazioni sui test in Device Farm, vedereFramework di test e test integrati in AWS Device Farm.

Argomenti

Crea i pacchetti per la tua XCTest corsa

Per testare la tua app utilizzando il XCTest framework, Device Farm richiede quanto segue:

- Il pacchetto dell'app come file `.ipa`.
- Il tuo XCTest pacchetto come `.zip` file.

È possibile creare questi pacchetti utilizzando l'output di compilazione generato da Xcode. Completa i seguenti passaggi per creare i pacchetti in modo da poterli caricare su Device Farm.

Per generare l'output di compilazione per l'app

1. Aprire il progetto di app in Xcode.
2. Nel menu a discesa dello schema nella barra degli strumenti di Xcode, scegliere Generic iOS Device (Dispositivo iOS generico) come destinazione.
3. Nel menu Product (Prodotto) scegliere Build For (Compilazione per) e selezionare Testing (Test).

Per creare il pacchetto di app

1. Nel navigatore di progetto in Xcode, aprire sotto Products (Prodotti) il menu contestuale per il file denominato *`app-project-name`*`.app`. Quindi, scegliere Show in Finder (Mostra in Finder).

Finder apre una cartella denominata `Debug-iphoneos`, che contiene l'output generato da Xcode per la compilazione di test. Questa cartella include il file `.app`.

2. In Finder, creare una nuova cartella e denominarla `Payload`.
3. Copiare il file *`app-project-name`*`.app` e incollarlo nella cartella `Payload`.
4. Aprire il menu contestuale per la cartella `Payload` e selezionare Compress "Payload" (Comprimi "Payload"). Viene creato un file denominato `Payload.zip`.
5. Sostituire il nome file e l'estensione di `Payload.zip` con *`app-project-name`*`.ipa`.

   In una fase successiva, fornirete questo file a Device Farm. Per rendere il file più facile da trovare, è possibile spostarlo in un'altra posizione, ad esempio sul desktop.
6. Facoltativamente, è possibile eliminare la cartella `Payload` e il file `.app` che contiene.

Per creare il XCTest pacchetto

1. In Finder, aprire nella directory `Debug-iphoneos` il menu contestuale per il file *`app-project-name`*`.app`. Quindi, scegliere Show Package Contents (Mostra contenuti pacchetto).
2. Nei contenuti del pacchetto, aprire la cartella `Plugins`. Questa cartella contiene un file denominato *`app-project-name`*`.xctest`.
3. Aprire il menu contestuale per questo file e scegliere Compress "***`app-project-name`*`.xctest`**" (Comprimi "app-project-name.xctest"). Viene creato un file denominato *`app-project-name`*`.xctest.zip`.

   In una fase successiva, fornirete questo file a Device Farm. Per rendere il file più facile da trovare, è possibile spostarlo in un'altra posizione, ad esempio sul desktop.

Carica i pacchetti per la tua XCTest corsa su Device Farm

Usa la console Device Farm per caricare i pacchetti per il test.

1. Accedere alla console Device Farm all'indirizzo https://console.aws.amazon.com/devicefarm.
2. Se non si dispone già di un progetto, creane uno. Per la procedura per creare un progetto, consultare Creazione di un progetto in AWS Device Farm.

   Altrimenti, nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.
3. Scegli il progetto che desideri utilizzare per eseguire il test.

4. Scegli Crea esegui.
5. In Impostazioni di esecuzione, nella sezione Tipo di esecuzione, scegli l'app iOS.
6. In Seleziona app, nella sezione Opzioni di selezione dell'app, seleziona Carica la tua app. Quindi, seleziona Scegli file in Carica app.
7. Individuare il file `.ipa` per l'app e caricarlo.

> **Note**
>
> Il pacchetto `.ipa` deve essere compilato per il test.

8. In Configura test, nella sezione Seleziona framework di test, scegli XCTest. Quindi, seleziona Scegli file in Carica app.
9. Cerca il `.zip` file che contiene il XCTest pacchetto per la tua app e caricalo.
10. Completare le fasi rimanenti del processo di creazione del progetto. Sarà quindi possibile selezionare i dispositivi che si desidera testare e specificare lo stato del dispositivo.
11. Scegli Crea esegui. Device Farm esegue il test e mostra i risultati nella console.

Integrazione dell' XCTest interfaccia utente per iOS con Device Farm

Device Farm fornisce supporto per il framework di test XCTest dell'interfaccia utente. In particolare, Device Farm supporta i test XCTest dell'interfaccia utente scritti sia in Objective-C che in Swift.

Il framework dell' XCTest interfaccia utente consente il test dell'interfaccia utente nello sviluppo di iOS, basato su XCTest. Per ulteriori informazioni, consulta la sezione User Interface Testing nella libreria per sviluppatori di iOS.

Per informazioni generali sui test in Device Farm, vedereFramework di test e test integrati in AWS Device Farm.

Utilizza le seguenti istruzioni per integrare Device Farm con il framework di test dell' XCTest interfaccia utente per iOS.

Argomenti

- Prepara i test XCTest dell'interfaccia utente iOS
- Opzione 1: creazione di un pacchetto XCTest UI .ipa
- Opzione 2: creazione di un pacchetto XCTest UI con estensione zip

- Carica i test XCTest dell'interfaccia utente iOS

Prepara i test XCTest dell'interfaccia utente iOS

Puoi caricare un `.ipa` file o un `.zip` file per il tuo pacchetto di test XCTEST_UI.

Un `.ipa` file è un archivio di applicazioni contenente l'app iOS Runner in formato bundle. Non è possibile includere file aggiuntivi all'interno del *`.ipa`* file.

Se carichi un `.zip` file, può contenere direttamente l'app iOS Runner o un `.ipa` file. Puoi anche includere altri file all'interno del `.zip` file se desideri utilizzarli durante i test. Ad esempio`.xctestrun`, puoi includere file come `.xcworkspace` o `.xcodeproj` all'interno `.zip` del file per eseguire i piani di test XCUI nella device farm. Istruzioni dettagliate su come eseguire i piani di test sono disponibili nel file delle specifiche di test predefinito per il tipo di test XCUI.

Opzione 1: creazione di un pacchetto XCTest UI .ipa

Il bundle yourAppNameUITest-Runner.app viene prodotto da Xcode quando crei il tuo progetto per i test. Si trova nella directory Products per il progetto.

Per creare un file.ipa:

1. Crea una directory chiamata. *`Payload`*
2. Aggiungi la directory dell'app alla directory Payload.
3. Archivia la directory Payload in un `.zip` file, quindi modifica l'estensione del file in. `.ipa`

La seguente struttura di cartelle mostra come un'app di esempio denominata *`my-project-nameUITest-Runner.app`* verrebbe impacchettata come `.ipa` file:

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

Opzione 2: creazione di un pacchetto XCTest UI con estensione zip

Device Farm genera automaticamente un `.xctestrun` file per l'esecuzione della suite di test XCTest dell'interfaccia utente completa. Se desideri utilizzare il tuo `.xctestrun` file su Device Farm,

puoi comprimere `.xctestrun` i file e la directory dell'app in un `.zip` file. Se hai già un `.ipa` file per il tuo pacchetto di test, puoi includerlo qui invece di*`*-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)
```

Se desideri eseguire un piano di test Xcode per i tuoi test XCUI su Device Farm, puoi creare uno zip contenente il file my-project-nameUITest-Runner.app o my-project-name UITest .ipa e i file di codice sorgente xcode necessari per eseguire XCTEST_UI con piani di test, incluso un file or. `.xcworkspace` `.xcodeproj`

Ecco un esempio di zip che utilizza un file: `.xcodeproj`

```
.
### 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)
```

Ecco un esempio di zip che utilizza un `.xcworkspace` file:

```
.
###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**
>
> Assicurati di non avere una directory denominata «Payload» all'interno del pacchetto XCTest UI .zip.

Carica i test XCTest dell'interfaccia utente iOS

Usa la console Device Farm per caricare i test.

1. Accedere alla console Device Farm all'indirizzo https://console.aws.amazon.com/devicefarm.
2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.
3. Nell'elenco dei progetti, scegli il progetto su cui caricare i test.

   > **Tip**
   >
   > Puoi utilizzare la barra di ricerca per filtrare l'elenco dei progetti per nome.
   > Per creare un progetto, segui le istruzioni in Creazione di un progetto in AWS Device Farm
4. Scegli Crea esegui.
5. In Impostazioni di esecuzione, nella sezione Tipo di esecuzione, scegli l'app iOS.
6. In Seleziona app, nella sezione Opzioni di selezione dell'app, seleziona Carica la tua app. Quindi, seleziona Scegli file in Carica app.
7. Individuare e selezionare il file dell'app iOS. Il file deve essere un file .ipa.

   > **Note**
   >
   > Assicurati che il file .ipa sia integrato per un dispositivo iOS e non per un simulatore.
8. In Configura test, nella sezione Seleziona framework di test, scegli XCTest UI. Quindi, seleziona Scegli file in Carica app.
9. Cerca e scegli il file.ipa o.zip che contiene il tuo test runner XCTest dell'interfaccia utente iOS.

10. Completa i passaggi rimanenti del processo di creazione dell'esecuzione. Selezionerai i dispositivi su cui desideri eseguire il test e, facoltativamente, specificherai una configurazione aggiuntiva.
11. Scegli Crea esegui. Device Farm esegue il test e mostra i risultati nella console.

Test di app Web in AWS Device Farm

Device Farm fornisce test con Appium per applicazioni web. Per ulteriori informazioni sulla configurazione dei test Appium su Device Farm, consulta. the section called “Test Appium automatici”

Per ulteriori informazioni sui test in Device Farm, vedereFramework di test e test integrati in AWS Device Farm.

Regole per i dispositivi misurati e non misurati

Per misurazione si intende la fatturazione per i dispositivi. Per impostazione predefinita, i dispositivi Device Farm vengono misurati e ti viene addebitato un addebito al minuto una volta esauriti i minuti di prova gratuiti. Puoi anche scegliere di acquistare dispositivi non misurati, per effettuare test illimitatamente a un costo mensile fisso. Per ulteriori informazioni sui prezzi, consulta la pagina dei prezzi di AWS Device Farm.

Se decidi di avviare una sessione con un pool di dispositivi che contiene sia dispositivi iOS che Android, sono previste regole per i dispositivi misurati e non misurati. Ad esempio, se sono presenti cinque dispositivi Android non misurati e cinque dispositivi iOS non misurati, le sessioni di test Web utilizzano i dispositivi non misurati.

Ecco un altro esempio: supponiamo di avere cinque dispositivi Android non misurati e 0 dispositivi iOS non misurati. Se selezioni solo i dispositivi Android per la sessione Web, vengono utilizzati i dispositivi non misurati. Se si selezionano entrambi i dispositivi Android e iOS per la sessione Web, il metodo di fatturazione viene misurato e i dispositivi non misurati non vengono utilizzati.

Test integrati in AWS Device Farm

Device Farm fornisce supporto per tipi di test integrati per dispositivi Android e iOS.

Grazie ai test integrati, puoi testare la tua applicazione su più dispositivi senza dover scrivere e gestire script di automazione dei test. Questo può farti risparmiare tempo e fatica, soprattutto quando inizi a usare Device Farm. Device Farm offre il seguente tipo di test integrato:

- Integrato: fuzz (Android e iOS)— Il fuzz test invia in modo casuale gli eventi dell'interfaccia utente ai dispositivi e quindi riporta i risultati.

Per ulteriori informazioni sui test e sui framework di test in Device Farm, vedere. Framework di test e test integrati in AWS Device Farm

Esecuzione del fuzz test integrato di Device Farm (Android e iOS)

Il fuzz test integrato di Device Farm invia in modo casuale gli eventi dell'interfaccia utente ai dispositivi e quindi riporta i risultati.

Per ulteriori informazioni sui test in Device Farm, vedereFramework di test e test integrati in AWS Device Farm.

Per eseguire il fuzz test integrato

1. Accedere alla console Device Farm all'indirizzo https://console.aws.amazon.com/devicefarm.
2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.
3. Nell'elenco dei progetti, scegli il progetto in cui desideri eseguire il fuzz test integrato.

   > **Tip**
   >
   > Puoi usare la barra di ricerca per filtrare l'elenco dei progetti per nome.
   > Per creare un progetto, segui le istruzioni indicate nella sezione Creazione di un progetto in AWS Device Farm.

4. Scegli Crea esegui.
5. In Impostazioni di esecuzione, seleziona il tipo di corsa nella sezione Tipo di corsa. Seleziona l'app per Android se non hai un'app pronta per il test o se stai testando un'app per Android (.apk). Seleziona App iOS se stai testando un'app iOS (.ipa).
6. In Seleziona app, scegli Seleziona app di esempio fornita da Device Farm se non hai un'app disponibile per il test. Se porti la tua app, seleziona Carica la tua app e scegli il file dell'applicazione.
7. In Configura test, nella sezione Seleziona framework di test, scegli Built-in: Fuzz.
8. Se compare una delle impostazioni seguenti, è possibile accettare i valori predefiniti o specificare i propri:

- Conteggio eventi: specificare un numero compreso tra 1 e 10.000, che rappresenta il numero di eventi di interfaccia utente che il test Fuzz deve eseguire.
- Event throttle: specifica un numero compreso tra 0 e 1.000, che rappresenta il numero di millisecondi di attesa del fuzz test prima di eseguire il successivo evento dell'interfaccia utente.
- Seed randomizer: specificare un numero che il test Fuzz utilizzerà per creare casualmente eventi di interfaccia utente. Se si specifica lo stesso numero per test Fuzz successivi, le sequenze di eventi saranno identiche.

9. Completa le istruzioni rimanenti per selezionare i dispositivi e avviare l'esecuzione.

Ambienti di test personalizzati in AWS Device Farm

AWS Device Farm consente di configurare un ambiente personalizzato per i test automatici (modalità personalizzata), che è l'approccio consigliato per tutti gli utenti di Device Farm. Per ulteriori informazioni sugli ambienti in Device Farm, consulta Ambienti di test.

I vantaggi della modalità personalizzata rispetto alla modalità standard includono:

- Esecuzione più rapida end-to-end dei test: il pacchetto di test non viene analizzato per rilevare tutti i test della suite, evitando il preprocessing/postprocessing sovraccarico.
- Registro live e streaming video: i registri dei test e i video sul lato client vengono trasmessi in live streaming quando si utilizza la modalità personalizzata. Questa funzionalità non è disponibile nella modalità standard.
- Cattura tutti gli artefatti: sull'host e sul dispositivo, la modalità personalizzata consente di acquisire tutti gli artefatti di test. Ciò potrebbe non essere possibile nella modalità standard.
- Ambiente locale più coerente e replicabile: in modalità standard, gli artefatti verranno forniti separatamente per ogni singolo test, il che può essere utile in determinate circostanze. Tuttavia, l'ambiente di test locale potrebbe differire dalla configurazione originale poiché Device Farm gestisce ogni test eseguito in modo diverso.

  Al contrario, la modalità personalizzata consente di rendere l'ambiente di esecuzione dei test di Device Farm costantemente in linea con l'ambiente di test locale.

Gli ambienti personalizzati sono configurati utilizzando un file di specifiche di test (test spec) in formato YAML. Device Farm fornisce un file di specifiche di test predefinito per ogni tipo di test supportato che può essere utilizzato così com'è o personalizzato; personalizzazioni come filtri di test o file di configurazione possono essere aggiunte alle specifiche di test. Le specifiche di test modificate possono essere salvate per future esecuzioni di test.

Per ulteriori informazioni, consulta Caricamento di una specifica di test personalizzata utilizzando and. AWS CLICreazione di un'esecuzione di test in Device Farm

Argomenti

- Riferimento e sintassi delle specifiche di test
- Host per ambienti di test personalizzati
- Accedi alle risorse AWS utilizzando un ruolo di esecuzione IAM

- Variabili di ambiente per ambienti di test personalizzati
- Le migliori pratiche per l'esecuzione di ambienti di test personalizzati
- Migrazione dei test da un ambiente di test standard a un ambiente di test personalizzato
- Estensione degli ambienti di test personalizzati in Device Farm

Riferimento e sintassi delle specifiche di test

La specifica di test (specifica del test) è un file che viene utilizzato per definire ambienti di test personalizzati in Device Farm.

Flusso di lavoro delle specifiche di test

La specifica di test di Device Farm esegue le fasi e i relativi comandi in un ordine predeterminato, consentendoti di personalizzare il modo in cui l'ambiente viene preparato ed eseguito. Quando ogni fase viene eseguita, i relativi comandi vengono eseguiti nell'ordine elencato nel file delle specifiche di test. Le fasi vengono eseguite nella seguente sequenza

1. `install`- È qui che devono essere definite azioni come il download, l'installazione e la configurazione degli strumenti.
2. `pre_test`- È qui che devono essere definite le azioni preliminari al test, come l'avvio di processi in background.
3. `test`- Qui deve essere definito il comando che richiama il test.
4. `post_test`- È qui che devono essere definite tutte le attività finali che devono essere eseguite dopo la conclusione del test, come la generazione di report di test e l'aggregazione di file di artefatti.

Sintassi delle specifiche del test

Di seguito è riportato lo schema YAML per un file di specifiche di test

```
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`

(Obbligatorio, numero)

Riflette la versione delle specifiche di test supportata da Device Farm. Il numero della versione corrente è `0.1`.

`android_test_host`

(Opzionale, stringa)

L'host di test che verrà selezionato per le esecuzioni di test eseguite su dispositivi Android. Questo campo è obbligatorio per i test eseguiti su dispositivi Android. Per ulteriori informazioni, consulta Host di test disponibili per ambienti di test personalizzati.

`ios_test_host`

(Opzionale, stringa)

L'host di test che verrà selezionato per le esecuzioni di test eseguite su dispositivi iOS. Questo campo è obbligatorio per le esecuzioni di test su dispositivi iOS con una versione principale

superiore a 26. Per ulteriori informazioni, consulta Host di test disponibili per ambienti di test personalizzati.

`phases`

Questa sezione contiene gruppi di comandi eseguiti durante un'esecuzione di test, in cui ogni fase è facoltativa. I nomi delle fasi di test consentiti sono: `install`,`pre_test`,`test`, e`post_test`.

- `install`- Le dipendenze predefinite per i framework di test supportati da Device Farm sono già installate. Questa fase contiene eventuali comandi aggiuntivi che Device Farm esegue durante l'installazione.
- `pre_test`- I comandi, se presenti, eseguiti prima del test automatico.
- `test`- I comandi eseguiti durante l'esecuzione del test automatico. Se un comando nella fase di test fallisce (il che significa che restituisce un codice di uscita diverso da zero), il test viene contrassegnato come fallito
- `post_test`- I comandi, se presenti, eseguiti dopo l'esecuzione del test automatico. Questo verrà eseguito indipendentemente dal fatto che il test nella `test` fase abbia esito positivo o negativo.

`commands`

(Facoltativo, List [string])

Un elenco di stringhe da eseguire come comando di shell durante la fase.

`artifacts`

(Facoltativo, List [string])

Device Farm raccoglie artefatti come report personalizzati, file di registro e immagini da una posizione specificata qui. I caratteri jolly non sono supportati come parte della posizione di un artefatto, perciò devi specificare un percorso valido per ogni posizione.

Questi artefatti di test sono disponibili per ciascun dispositivo nella tua sessione di test. Per informazioni sul recupero degli artefatti di test, consulta Scaricamento di artefatti in un ambiente di test personalizzato.

> **Important**
>
> Una specifica di test deve essere formattata come file YAML valido. Se l'indentazione o la spaziatura della tua specifica di test non è valida, il tuo test può non riuscire. Le schede

non sono consentite nei file YAML. Puoi usare un validatore YAML per verificare se la tua specifica di test è un file YAML valido. Per ulteriori informazioni, consulta il sito Web YAML.

Esempi di specifiche di test

Gli esempi seguenti mostrano le specifiche di test che possono essere eseguite su Device Farm.

Simple Demo

Di seguito è riportato un esempio di file di specifiche di test che viene semplicemente registrato `Hello world!` come elemento di test run.

```
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

Di seguito è riportato un esempio di file di test spec che configura un test Appium Java TestNG eseguito su Android..

```
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
```

Appio iOS

Di seguito è riportato un esempio di file di test spec che configura un test Appium Java TestNG eseguito su iOS.

```
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)

Di seguito è riportato un esempio di file di test spec che configura un test Appium Java TestNG eseguito su Android e iOS.

```
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
```

Host per ambienti di test personalizzati

Device Farm supporta una serie di sistemi operativi con software preconfigurato tramite l'uso di un ambiente host di test. Durante l'esecuzione del test, Device Farm utilizza istanze (host) gestite da Amazon che si connettono dinamicamente al dispositivo selezionato in fase di test. Questa istanza viene completamente ripulita e non riutilizzata tra un'esecuzione e l'altra e viene terminata con gli artefatti generati al termine dell'esecuzione del test.

Argomenti

- Host di test disponibili per ambienti di test personalizzati
- Selezione di un host di test per ambienti di test personalizzati
- Software supportato in ambienti di test personalizzati

- Ambiente di test per dispositivi Android
- Ambiente di test per dispositivi iOS

Host di test disponibili per ambienti di test personalizzati

Gli host di test sono completamente gestiti da Device Farm. La tabella seguente elenca gli host di test Device Farm attualmente disponibili e supportati per ambienti di test personalizzati.

Piattaforma del dispositivo	Host di test	Sistema operativo	Architettura/e	Dispositivi supportati
Android	amazon_linux_2	Amazon Linux 2	x86_64	Android6 e versioni successive
iOS	macos_sequoia	macOS Sequoia(versione 15)	arm64	iOSda 15 a 26

> **Note**
>
> Periodicamente, Device Farm aggiunge nuovi host di test per una piattaforma di dispositivi per supportare le versioni più recenti del sistema operativo del dispositivo e le relative dipendenze. Quando ciò si verifica, gli host di test precedenti per la rispettiva piattaforma di dispositivi sono soggetti alla fine del supporto.

Versione del sistema operativo

Ogni host di test disponibile utilizza una versione specifica del sistema operativo supportata in quel momento da Device Farm. Sebbene cerchiamo di utilizzare la versione più recente del sistema operativo, questa potrebbe non essere l'ultima versione distribuita pubblicamente disponibile. Device Farm aggiornerà periodicamente il sistema operativo con aggiornamenti di versione minori e patch di sicurezza.

Per conoscere la versione specifica (inclusa la versione secondaria) del sistema operativo in uso durante l'esecuzione del test, puoi aggiungere il seguente frammento di codice a qualsiasi fase del file delle specifiche di test.

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
```

Selezione di un host di test per ambienti di test personalizzati

Puoi specificare l'host di test Android e iOS nelle `ios_test_host` variabili appropriate `android_test_host` del file delle specifiche di test.

Se non si specifica una selezione di host di test per una determinata piattaforma di dispositivi, i test verranno eseguiti sull'host di test che Device Farm ha impostato come predefinito per il dispositivo e la configurazione di test specificati.

> **Important**
>
> Durante i test su iOS 18 e versioni precedenti, verrà utilizzato un host di test legacy quando non è selezionato un host. Per ulteriori informazioni, consulta l'argomento suHost di test iOS legacy.

Ad esempio, esamina il seguente frammento di codice:

Example

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

phases:
  # ...
```

Software supportato in ambienti di test personalizzati

Device Farm utilizza macchine host preinstallate con molte delle librerie software necessarie per eseguire i framework di test supportati dal nostro servizio, fornendo un ambiente di test pronto all'avvio. Device Farm supporta più lingue tramite l'uso del nostro meccanismo di selezione del software e aggiornerà periodicamente le versioni delle lingue incluse nell'ambiente.

Per qualsiasi altro software richiesto, puoi modificare il file delle specifiche di test per installarlo dal tuo pacchetto di test, scaricarlo da Internet o accedere a fonti private all'interno del tuo VPC (vedi VPC ENI per ulteriori informazioni). Per ulteriori informazioni, consulta Esempi di specifiche di test.

Software preconfigurato

Per facilitare il test dei dispositivi su ciascuna piattaforma, sull'host di test sono disponibili i seguenti strumenti:

Tools (Strumenti)	Piattaforme per dispositivi
Android SDK Build-Tools	Android
Android SDK Platform-Tools(includeadb)	Android
Xcode	iOS

Software selezionabile

Oltre al software preconfigurato sull'host, Device Farm offre un modo per selezionare determinate versioni del software supportato tramite gli `devicefarm-cli` strumenti.

La tabella seguente contiene il software selezionabile e gli host di test che li contengono.

Software/Strumento	Host che supportano questo software	Comando da utilizzare nelle specifiche di test
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`
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`

Software/Strumento	Host che supportano questo software	Comando da utilizzare nelle specifiche di test
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`

L'host di test include anche strumenti di supporto di uso comune per ogni versione del software, come i `pip` gestori di `npm` pacchetti (inclusi rispettivamente in Python e Node.js) e le dipendenze (come il UIAutomator2 driver Appium) per strumenti come Appium. Ciò garantisce di disporre degli strumenti necessari per lavorare con i framework di test supportati.

Utilizzo dello strumento devicefarm-cli in ambienti di test personalizzati

L'host di test utilizza uno strumento di gestione delle versioni standardizzato chiamato a selezionare le versioni del software. `devicefarm-cli` Questo strumento è separato dall'host di test Device Farm AWS CLI ed è disponibile solo sull'host di test Device Farm. Con`devicefarm-cli`, è possibile passare a qualsiasi versione software preinstallata sull'host di test. Ciò offre un modo semplice per mantenere il file delle specifiche di test di Device Farm nel tempo e offre un meccanismo prevedibile per aggiornare le versioni del software in futuro.

> **Important**
>
> Questo strumento da riga di comando non è disponibile sugli host iOS precedenti. Per ulteriori informazioni, consulta l'argomento suHost di test iOS legacy.

Il frammento seguente mostra la `help` pagina di: `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.
```

Esaminiamo un paio di esempi di utilizzo di. `devicefarm-cli` Per utilizzare lo strumento per modificare la versione di Python da *`3.10`* a *`3.9`* nel file delle specifiche di test, esegui i seguenti comandi:

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

Per cambiare la versione di Appium da a: *`1`* *`2`*

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

> **Tip**
>
> Nota che quando selezioni una versione del software, cambia `devicefarm-cli` anche gli strumenti di supporto per quei linguaggi, come `pip` Python `npm` e NodeJS.

Per ulteriori informazioni sul software preinstallato sull'host di test, consulta. Software supportato in ambienti di test personalizzati

Ambiente di test per dispositivi Android

AWS Device Farm utilizza macchine host Amazon Elastic Compute Cloud (EC2) che eseguono Amazon Linux 2 per eseguire test Android. Quando pianifichi un'esecuzione di test, Device Farm assegna un host dedicato per ciascun dispositivo per eseguire i test in modo indipendente. Le macchine host si interrompono dopo l'esecuzione del test insieme a tutti gli artefatti generati.

L'host Amazon Linux 2 offre diversi vantaggi:

- Test più veloci e affidabili: rispetto all'host precedente, il nuovo host di test migliora significativamente la velocità dei test, in particolare riducendo i tempi di inizio dei test. L'host Amazon Linux 2 dimostra inoltre una maggiore stabilità e affidabilità durante i test.
- Accesso remoto avanzato per i test manuali: gli aggiornamenti all'host di test più recente e i miglioramenti comportano una minore latenza e migliori prestazioni video per i test manuali Android.
- Selezione della versione software standard: Device Farm ora standardizza il supporto dei principali linguaggi di programmazione sull'host di test e sulle versioni del framework Appium. Per i linguaggi supportati (attualmente Java, Python, Node.js e Ruby) e Appium, il nuovo host di test fornisce versioni stabili a lungo termine subito dopo il lancio. La gestione centralizzata delle versioni tramite `devicefarm-cli` lo strumento consente lo sviluppo di file con specifiche di test con un'esperienza coerente tra i framework.

Argomenti

- Intervalli IP supportati per l'ambiente di test Amazon Linux 2 in Device Farm

Intervalli IP supportati per l'ambiente di test Amazon Linux 2 in Device Farm

I clienti spesso hanno bisogno di conoscere l'intervallo di IP da cui proviene il traffico di Device Farm, in particolare per configurare i firewall e le impostazioni di sicurezza. Per gli host di test Amazon EC2, l'intervallo IP comprende l'intera regione. `us-west-2` Per gli host di test di Amazon Linux 2, che è l'opzione predefinita per le nuove esecuzioni di Android, gli intervalli sono stati limitati. Il traffico ora proviene da un set specifico di gateway NAT, limitando l'intervallo IP ai seguenti indirizzi:

Intervalli IP
44.236.137,143
5213,151,44
5235,189,1991
54201.250,26

Per ulteriori informazioni sugli ambienti di test Android in Device Farm, consultaAmbiente di test per dispositivi Android.

Ambiente di test per dispositivi iOS

Device Farm utilizza istanze macOS (host) gestite da Amazon che si connettono dinamicamente al dispositivo iOS durante l'esecuzione del test. Ogni host è preconfigurato con un software che consente il test dei dispositivi su varie piattaforme di test popolari, come UI e Appium. XCTest

L'attuale iterazione dell'host di test iOS ha migliorato l'esperienza di test rispetto alle versioni precedenti, tra cui:

- Sistema operativo host e strumenti coerenti per iOS 15 e iOS 26 In precedenza, l'host di test veniva determinato dal dispositivo in uso, il che portava a un ambiente software frammentato durante l'esecuzione su più versioni di iOS. L'esperienza attuale consente una semplice selezione dell'host per consentire un ambiente coerente su tutti i dispositivi. Ciò consentirà di rendere disponibili la stessa versione e gli stessi strumenti di macOS (come Xcode) su ogni dispositivo iOS.
- Miglioramenti delle prestazioni per i test di iOS 15 e 16 Utilizzando un'infrastruttura aggiornata, i tempi di configurazione sono notevolmente migliorati per i test di iOS 15 e 16.

- Versioni software selezionabili standardizzate per le dipendenze supportate Ora disponiamo del sistema di selezione del `devicefarm-cli` software sugli host di test iOS e Android, che consente di selezionare la versione preferita delle nostre dipendenze supportate. Per le dipendenze supportate (come Java, Python, Node.js, Ruby e Appium), le versioni saranno selezionabili tramite le specifiche di test. Per un'idea di come funziona questa funzionalità, consulta l'argomento su. Software supportato in ambienti di test personalizzati

> **Important**
>
> Se esegui su iOS 18 e versioni precedenti, i test verranno eseguiti su host di test legacy per impostazione predefinita. Consulta l'argomento seguente su come migrare da host legacy.

Host di test iOS legacy

Per i test esistenti su iOS 18 e versioni precedenti, gli host di test legacy sono selezionati per impostazione predefinita per gli ambienti di test personalizzati. La tabella seguente contiene la versione dell'host di test eseguita con la versione del dispositivo iOS.

Sistema operativo	Architettura/e	Impostazione predefinita per i dispositivi
macOS Sonoma(versione 14)	arm64	iOS 18
macOS Ventura(versione 13)	arm64	iOS 17
macOS Monterey(versione 12)	x86_64	iOS 16e di seguito

Per selezionare gli host di test più recenti, consulta l'argomento relativoMigrazione degli ambienti di test personalizzati ai nuovi host di test iOS.

Software supportato per dispositivi iOS

Per supportare i test dei dispositivi iOS, gli host di test Device Farm per dispositivi iOS sono preconfigurati con Xcode e gli strumenti a riga di comando associati. Per altri software disponibili, consulta l'argomento relativo. Software supportato in ambienti di test personalizzati

Migrazione degli ambienti di test personalizzati ai nuovi host di test iOS

Per migrare i test esistenti dall'host legacy al nuovo host di test macOS, dovrai sviluppare nuovi file di specifiche di test basati su quelli preesistenti.

L'approccio consigliato consiste nell'iniziare con il file di test spec di esempio per i tipi di test desiderati, quindi migrare i comandi pertinenti dal vecchio file delle specifiche di test a quello nuovo. Ciò consente di sfruttare le nuove funzionalità e ottimizzazioni delle specifiche di test di esempio per il nuovo host riutilizzando al contempo gli snippet del codice esistente.

Argomenti

Tutorial: migrazione dei file delle specifiche di test iOS con la console

In questo esempio, la console Device Farm verrà utilizzata per integrare le specifiche di test di un dispositivo iOS esistente per utilizzare il nuovo host di test.

Passaggio 1: creazione di nuovi file delle specifiche di test con la console

1. Accedi alla console AWS Device Farm.
2. Vai al progetto Device Farm contenente i tuoi test di automazione.
3. Scarica una copia delle specifiche di test esistenti che desideri utilizzare.
    a. Fai clic sull'opzione «Impostazioni del progetto» e vai alla scheda Caricamenti.
    b. Vai al file delle specifiche di test con cui desideri effettuare l'onboarding.
    c. Fai clic sul pulsante Download per creare una copia locale di questo file.
4. Torna alla pagina del progetto e fai clic su Crea esegui.
5. Compila le opzioni della procedura guidata come se dovessi iniziare una nuova esecuzione, ma fermati all'opzione Select test spec.
6. Utilizzando la specifica di test iOS selezionata per impostazione predefinita, fai clic sul pulsante Crea una specifica di test.
7. Modifica la specifica del test selezionata per impostazione predefinita nell'editor di testo.
    a. Se non è già presente, modifica il file delle specifiche di test per selezionare il nuovo host utilizzando:

```
ios_test_host: macos_sequoia
```

b. Dalla copia delle specifiche di test scaricata in un passaggio precedente, esaminale ciascuna. `phase`

c. Copia i comandi dalle fasi della vecchia specifica di test in ciascuna rispettiva fase della nuova specifica di test, ignorando i comandi relativi all'installazione o alla selezione di Java, Python, Node.js, Ruby, Appium o Xcode.

8. Immettete un nuovo nome di file nella casella di testo Salva come.
9. Fai clic sul pulsante Salva come nuovo per salvare le modifiche.

Per un esempio di file di specifiche di test che puoi usare come riferimento, vedi l'esempio fornito inEsempi di specifiche di test.

Fase 2: Selezione del software preinstallato

Nel nuovo host di test, le versioni software preinstallate vengono selezionate utilizzando un nuovo strumento di gestione delle versioni standardizzato chiamato. `devicefarm-cli` Questo strumento è ora l'approccio consigliato per l'utilizzo dei vari software che forniamo sugli host di test.

Ad esempio, dovresti aggiungere la riga seguente per utilizzare un JDK 17 diverso nel tuo ambiente di test:

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

Per ulteriori informazioni sul software supportato disponibile, consulta:. Software supportato in ambienti di test personalizzati

Passo 3: Utilizzo di Appium e delle sue dipendenze tramite gli strumenti di selezione del software

Il nuovo host di test supporta solo Appium 2.x e versioni successive. Seleziona esplicitamente la versione di Appium utilizzando`devicefarm-cli`, rimuovendo strumenti legacy come. `avm` Esempio:

```
# 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
```

La versione Appium selezionata con `devicefarm-cli` viene preinstallata con una versione compatibile del driver XCUITest per iOS.

Inoltre, dovrai aggiornare le specifiche del test per utilizzare invece di. `DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V9` `DEVICEFARM_WDA_DERIVED_DATA_PATH` La nuova variabile di ambiente punta a una versione predefinita di WebDriverAgent 9.x, che è l'ultima versione supportata per i test di Appium 2.

Per ulteriori informazioni, consulta e. Selezione di una WebDriverAgent versione per i test iOS Variabili di ambiente per i test di Appium

Differenze tra gli host di test nuovi e quelli precedenti

Quando modifichi il file delle specifiche di test per utilizzare il nuovo host di test iOS e trasferisci i test dall'host di test precedente, tieni presente queste differenze chiave nell'ambiente:

- Versioni Xcode: nell'ambiente host di test legacy, la versione Xcode disponibile era basata sulla versione iOS del dispositivo utilizzato per i test. Ad esempio, i test su dispositivi iOS 18 hanno utilizzato Xcode 16 nell'host legacy, mentre i test su iOS 17 hanno utilizzato Xcode 15. Nel nuovo ambiente host, tutti i dispositivi possono accedere alle stesse versioni di Xcode, garantendo un ambiente coerente per i test su dispositivi con versioni diverse. Per un elenco delle versioni di Xcode attualmente disponibili, consulta. Software supportato
- Selezione delle versioni del software: in molti casi, le versioni software predefinite sono cambiate, quindi se prima non selezionavi esplicitamente la versione del software nell'host di test precedente, potresti volerla specificare ora nel nuovo host di test che utilizza. `devicefarm-cli` Nella maggior parte dei casi d'uso, consigliamo ai clienti di selezionare esplicitamente le versioni del software che utilizzano. Selezionando una versione del software con, `devicefarm-cli` avrai un'esperienza prevedibile e coerente e riceverai un'ampia quantità di avvisi se Device Farm prevede di rimuovere quella versione dall'host di test.

  Inoltre, strumenti di selezione del software come `nvm` `pyenv` `avm`, e `rvm` sono stati rimossi a favore del nuovo `devicefarm-cli` sistema di selezione del software.
- Versioni software disponibili: molte versioni del software preinstallato in precedenza sono state rimosse e sono state aggiunte molte nuove versioni. Pertanto, assicurati che quando utilizzi `devicefarm-cli` per selezionare le versioni del software, selezioni le versioni presenti nell'elenco delle versioni supportate.

- La **`libimobiledevice`** suite di strumenti è stata rimossa a favore di strumenti più nuovi/di prima parte per tenere traccia degli attuali test dei dispositivi iOS e degli standard di settore. Per iOS 17 e versioni successive, puoi migrare la maggior parte dei comandi per utilizzare strumenti Xcode simili, chiamati. `devicectl` Per informazioni in meritodevicectl, puoi eseguirlo `xcrun devicectl help` da una macchina su cui è installato Xcode.
- I percorsi di file codificati nel file delle specifiche di test dell'host precedente come percorsi assoluti molto probabilmente non funzioneranno come previsto nel nuovo host di test e generalmente non sono consigliati per l'uso di file di specifiche di test. Ti consigliamo di utilizzare percorsi relativi e variabili di ambiente per tutto il codice del file delle specifiche di test. Per ulteriori informazioni, consulta l'argomento suLe migliori pratiche per l'esecuzione di ambienti di test personalizzati.
- Versione e architettura del sistema operativo: gli host di test precedenti utilizzavano una varietà di versioni macOS e architetture CPU in base al dispositivo assegnato. Di conseguenza, gli utenti potrebbero notare alcune differenze nelle librerie di sistema disponibili nell'ambiente. Per ulteriori informazioni sulla versione precedente del sistema operativo host, consultaHost di test iOS legacy.
- Per gli utenti di Appium, il modo di selezionare WebDriverAgent è stato modificato in un prefisso per le variabili di ambiente utilizzate `DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V` anziché il vecchio prefisso. `DEVICEFARM_WDA_DERIVED_DATA_PATH_V` Per ulteriori informazioni sulla variabile aggiornata, consulta. Variabili di ambiente per i test di Appium
- Per gli utenti di Appium Java, il nuovo host di test non contiene alcun file JAR preinstallato nel percorso di classe, mentre l'host precedente ne conteneva uno per il framework TestNG (tramite una variabile di ambiente). `$DEVICEFARM_TESTNG_JAR` Consigliamo ai clienti di impacchettare i file JAR necessari per i propri framework di test all'interno del pacchetto di test e di rimuovere le istanze della variabile dai file delle `$DEVICEFARM_TESTNG_JAR` specifiche di test.

Ti consigliamo di contattare il team di assistenza tramite un caso di supporto se hai commenti o domande sulle differenze tra gli host di test dal punto di vista del software.

Accedi alle risorse AWS utilizzando un ruolo di esecuzione IAM

Device Farm supporta la specifica di un ruolo IAM che verrà assunto dall'ambiente di esecuzione del test personalizzato durante l'esecuzione del test. Questa funzionalità consente ai test di accedere in modo sicuro alle risorse AWS nel tuo account, come bucket Amazon S3, tabelle DynamoDB o altri servizi AWS da cui dipende l'applicazione.

Argomenti

- Panoramica

Panoramica

Quando si specifica un ruolo di esecuzione IAM, Device Farm lo assume durante l'esecuzione del test, consentendo ai test di interagire con i servizi AWS utilizzando le autorizzazioni definite nel ruolo.

I casi d'uso comuni per i ruoli di esecuzione IAM includono:

- Accesso ai dati di test archiviati nei bucket Amazon S3
- Trasferimento degli artefatti di test ai bucket Amazon S3
- Recupero della configurazione dell'applicazione da AWS AppConfig
- Scrittura di log e metriche dei test su Amazon CloudWatch
- Invio di risultati di test o messaggi di stato alle code di Amazon SQS
- Richiamo delle funzioni AWS Lambda come parte dei flussi di lavoro di test

Requisiti del ruolo IAM

Per utilizzare un ruolo di esecuzione IAM con Device Farm, il ruolo deve soddisfare i seguenti requisiti:

- Relazione di fiducia: è necessario affidare il ruolo al responsabile del servizio Device Farm. La politica di fiducia deve essere inclusa `devicefarm.amazonaws.com` come entità attendibile.
- Autorizzazioni: il ruolo deve disporre delle autorizzazioni necessarie per accedere alle risorse AWS richieste dai test.
- Durata della sessione: la durata massima della sessione del ruolo deve essere almeno pari all'impostazione del timeout del lavoro del progetto Device Farm. Per impostazione predefinita, i progetti Device Farm hanno un timeout di lavoro di 150 minuti, quindi il tuo ruolo deve supportare una sessione della durata di almeno 150 minuti.
- Stesso requisito dell'account: il ruolo IAM deve trovarsi nello stesso account AWS utilizzato per chiamare Device Farm. L'assunzione di ruoli tra account non è supportata.

- PassRole autorizzazione: il chiamante deve essere autorizzato a passare il ruolo IAM mediante una policy che consenta l'`iam:PassRole`azione sul ruolo di esecuzione specificato.

Esempio di policy di attendibilità

L'esempio seguente mostra una politica di attendibilità che consente a Device Farm di assumere il ruolo di esecuzione dell'utente. Questa policy di fiducia deve essere associata solo al ruolo IAM specifico che intendi utilizzare con Device Farm, non ad altri ruoli nel tuo account:

Example

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

Policy di autorizzazione di esempio

L'esempio seguente mostra una politica di autorizzazioni che garantisce l'accesso ai servizi AWS comuni utilizzati nei test:

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-*"
    }
  ]
}
```

Configurazione di un ruolo di esecuzione IAM

Puoi specificare un ruolo di esecuzione IAM a livello di progetto o per singole esecuzioni di test. Se configurate a livello di progetto, tutte le esecuzioni all'interno di quel progetto erediteranno il ruolo di esecuzione. Un ruolo di esecuzione configurato in un'esecuzione sostituirà qualsiasi ruolo configurato nel relativo progetto principale.

Per istruzioni dettagliate sulla configurazione dei ruoli di esecuzione, consulta:

- Creazione di un progetto in AWS Device Farm- per configurare i ruoli di esecuzione a livello di progetto
- Creazione di un'esecuzione di test in Device Farm- per configurare i ruoli di esecuzione per singole esecuzioni

Puoi anche configurare i ruoli di esecuzione utilizzando l'API Device Farm. Per ulteriori informazioni, consulta il Device Farm API Reference.

Best practice

Segui queste best practice per configurare i ruoli di esecuzione IAM per i test di Device Farm:

- Principio del privilegio minimo: concedi solo le autorizzazioni minime necessarie per il funzionamento dei test. Evita di utilizzare autorizzazioni troppo ampie come * azioni o risorse.
- Utilizza autorizzazioni specifiche per le risorse: quando possibile, limita le autorizzazioni a risorse specifiche (ad esempio, bucket S3 specifici o tabelle DynamoDB) anziché a tutte le risorse di un tipo.
- Risorse di test e produzione separate: utilizza risorse e ruoli di test dedicati per evitare di influire accidentalmente sui sistemi di produzione durante i test.
- Revisione periodica dei ruoli: rivedi e aggiorna periodicamente i ruoli di esecuzione per assicurarti che soddisfino ancora le tue esigenze di test e seguano le migliori pratiche di sicurezza.
- Usa le chiavi di condizione: valuta la possibilità di utilizzare le chiavi di condizione IAM per limitare ulteriormente quando e come il ruolo può essere utilizzato.

risoluzione dei problemi

Se riscontri problemi con i ruoli di esecuzione IAM, controlla quanto segue:

- Relazione di fiducia: verifica che la politica di fiducia del `devicefarm.amazonaws.com` ruolo includa un servizio affidabile.
- Autorizzazioni: verifica che il ruolo disponga delle autorizzazioni necessarie per i servizi AWS a cui i test stanno tentando di accedere.
- Registri di test: esamina i log di esecuzione dei test per messaggi di errore specifici relativi alle chiamate API AWS o ai dinieghi di autorizzazione.

Variabili di ambiente per ambienti di test personalizzati

Device Farm configura dinamicamente diverse variabili di ambiente da utilizzare come parte dell'esecuzione dell'ambiente di test personalizzato.

Argomenti

Variabili di ambiente personalizzate

Device Farm supporta la configurazione di coppie chiave-valore che vengono applicate come variabili di ambiente sull'host di test. Queste possono essere configurate in un progetto Device Farm o durante la creazione dell'esecuzione; tutte le variabili configurate in un'esecuzione sostituiranno quelle che possono essere configurate nel relativo progetto principale. Le restrizioni si applicano come segue:

- Le variabili di ambiente personalizzate non sono supportate sugli host di test iOS precedenti. Per ulteriori informazioni, consulta Host di test iOS legacy.
- I nomi delle variabili che iniziano con `$DEVICEFARM_` sono riservati all'uso interno del servizio.
- Le variabili di ambiente personalizzate non possono essere utilizzate per configurare la selezione del calcolo dell'host di test nelle specifiche del test.

Variabili di ambiente comuni

Questa sezione descrive le variabili di ambiente comuni a tutti i test in Device Farm.

$DEVICEFARM_DEVICE_NAME

Il dispositivo su cui vengono eseguiti i test. Rappresenta l'identificatore univoco del dispositivo (UDID).

$DEVICEFARM_DEVICE_UDID

L'identificatore univoco del dispositivo.

$DEVICEFARM_DEVICE_PLATFORM_NAME

Il nome della piattaforma del dispositivo. O è `Android` o `iOS`.

$DEVICEFARM_DEVICE_OS_VERSION

La versione del sistema operativo del dispositivo.

$DEVICEFARM_APP_PATH

(test delle app per dispositivi mobili)

Il percorso all'app mobile sulla macchina host dove vengono eseguiti i test. Questa variabile non è disponibile durante i test web.

$DEVICEFARM_LOG_DIR

Il percorso della directory predefinita in cui verranno archiviati i log del cliente, gli artefatti e gli altri file desiderati per un successivo recupero. Utilizzando una specifica di test di esempio, i file in questa directory vengono archiviati in un file ZIP e resi disponibili come artefatto dopo l'esecuzione del test.

$DEVICEFARM_SCREENSHOT_PATH

Il percorso agli screenshot, se esistenti, acquisiti durante la sessione di test.

$DEVICEFARM_PROJECT_ARN

L'ARN del progetto principale del lavoro.

$DEVICEFARM_RUN_ARN

L'ARN dell'esecuzione principale del processo.

$DEVICEFARM_DEVICE_ARN

L'ARN del dispositivo sottoposto a test.

$DEVICEFARM_TOTAL_JOBS

Il numero totale di job associati all'esecuzione della Device Farm principale.

$DEVICEFARM_JOB_NUMBER

Il numero di questo lavoro è compreso$DEVICEFARM_TOTAL_JOBS. Ad esempio, un'esecuzione può contenere 5 job, ognuno dei quali avrà un valore univoco $DEVICEFARM_JOB_NUMBER compreso tra 0 e 4.

$AWS_REGION

La regione AWS. Il servizio lo imposterà in modo che corrisponda alla regione in cui si trova il dispositivo sottoposto a test. Se necessario, può essere sovrascritto da una variabile di ambiente personalizzata.

$ANDROID_HOME

(Solo Android)

Il percorso della directory di installazione di Android SDK.

Variabili di ambiente per i test di Appium

Questa sezione descrive le variabili di ambiente utilizzate da qualsiasi test Appium in un ambiente di test personalizzato in Device Farm.

$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR

(Solo Android)

La posizione di una directory che contiene gli ChromeDriver eseguibili necessari per l'uso nei test web e ibridi di Appium.

$DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V<N>

(solo iOS)

Il percorso dati derivato di una versione di Device Farm WebDriverAgent creata per essere eseguita su Device Farm. La numerazione sulla variabile corrisponderà alla versione principale di WebDriverAgent. Ad esempio, DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V9 indicherà la WebDriverAgent versione a 9.x. Per ulteriori informazioni, consulta Selezione di una WebDriverAgent versione per i test iOS.

> **Note**
>
> Le variabili di `$DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V<N>` ambiente sono presenti solo su host iOS non legacy. Per ulteriori informazioni, consulta Host di test iOS legacy.

`$DEVICEFARM_WDA_DERIVED_DATA_PATH_V9`

(solo iOS, obsoleto)

Il percorso dati derivato di una versione di Device Farm WebDriverAgent creata per essere eseguita su Device Farm. `$DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V<N>`Per lo schema di denominazione sostitutivo, fare riferimento a.

Variabili di ambiente per i test XCUITest

Questa sezione descrive le variabili di ambiente utilizzate dal XCUITest test in un ambiente di test personalizzato in Device Farm.

`$DEVICEFARM_XCUITESTRUN_FILE`

Il percorso del `.xctestun` file Device Farm. Viene generato dalla tua app e dai pacchetti di test.

`$DEVICEFARM_DERIVED_DATA_PATH`

Percorso previsto dell'output xcodebuild di Device Farm.

`$DEVICEFARM_XCTEST_BUILD_DIRECTORY`

Il percorso ai contenuti decompressi del file del pacchetto di test.

Le migliori pratiche per l'esecuzione di ambienti di test personalizzati

Negli argomenti seguenti vengono illustrate le best practice consigliate per l'utilizzo dell'esecuzione di test personalizzati con Device Farm.

Configurazione run

- Affidati al software gestito da Device Farm e alle funzionalità API per eseguire la configurazione laddove possibile, anziché applicare configurazioni simili tramite comandi shell nel file delle specifiche di test. Ciò include la configurazione dell'host di test e del dispositivo, in quanto sarà più sostenibile e coerente tra host e dispositivi di test.

  Sebbene Device Farm vi incoraggi a personalizzare il file delle specifiche di test nella misura necessaria per eseguire i test, il file delle specifiche di test può diventare difficile da mantenere nel tempo man mano che vi vengono aggiunti comandi più personalizzati. Utilizzando il software gestito da Device Farm (tramite strumenti come `devicefarm-cli` e gli strumenti predefiniti disponibili in`$PATH`) e utilizzando funzionalità gestite (come il parametro `deviceProxy`request) per semplificare il file delle specifiche di test spostando la responsabilità della manutenzione su Device Farm stessa.

Specifiche di test e codice del pacchetto di test

- Non utilizzate percorsi assoluti né fate affidamento su versioni secondarie specifiche nel file delle specifiche di test o nel codice del pacchetto di test. Device Farm applica gli aggiornamenti di routine all'host di test selezionato e alle versioni software incluse. L'utilizzo di percorsi specifici o assoluti (ad esempio « `/usr/local/bin/python`instead of»`python`) o la richiesta di versioni secondarie specifiche (ad esempio Node.js `20.3.1` anziché «just» `20`) possono impedire ai test di individuare il file eseguibile/richiesto.

  Come parte dell'esecuzione dei test personalizzati, Device Farm imposta varie variabili di ambiente e la `$PATH` variabile per garantire che i test abbiano un'esperienza coerente all'interno dei nostri ambienti dinamici. Per ulteriori informazioni, consulta Variabili di ambiente per ambienti di test personalizzati e Software supportato in ambienti di test personalizzati.
- Salva i file generati o copiati nella directory temp durante l'esecuzione del test. Oggi ci assicuriamo che la directory temporanea (`/tmp`) sia accessibile all'utente durante l'esecuzione del test (oltre alle directory gestite, come la). `$DEVICEFARM_LOG_DIR` Le altre directory a cui l'utente ha accesso possono cambiare nel tempo a causa delle esigenze del servizio o del sistema operativo in uso.
- Salva i registri di esecuzione dei test in. **`$DEVICEFARM_LOG_DIR`** Questa è la directory degli artefatti predefinita fornita per l'esecuzione in cui aggiungere i log di esecuzione /gli artefatti. Le specifiche di test di esempio che forniamo utilizzano ciascuna questa directory per gli artefatti per impostazione predefinita.

- Assicurati che i tuoi comandi restituiscano un codice diverso da zero in caso di errore durante la `test` fase delle specifiche di test. Determiniamo se l'esecuzione è fallita controllando la presenza di un codice di uscita diverso da zero per ogni comando di shell invocato durante la fase. `test` È necessario assicurarsi che la logica o il framework di test restituiscano un codice di uscita diverso da zero per tutti gli scenari desiderati, il che potrebbe richiedere una configurazione aggiuntiva.

  Ad esempio, alcuni framework di test (come JUnit5) non considerano l'esecuzione di zero test un errore, il che determinerà la corretta esecuzione dei test anche se non è stato eseguito nulla. Ad esempio, è necessario specificare l'opzione della riga di comando per garantire che questo scenario `--fail-if-no-tests` venga chiuso con un codice di uscita diverso da zero. JUnit5
- Verifica la compatibilità del software con la versione del sistema operativo del dispositivo e la versione dell'host di test che utilizzerai per l'esecuzione del test. Ad esempio, alcune funzionalità di test dei framework software (ad esempio: Appium) potrebbero non funzionare come previsto su tutte le versioni del sistema operativo del dispositivo in fase di test.

Sicurezza

- Evita di archiviare o registrare variabili sensibili (come le chiavi AWS) nel file delle specifiche di test. I file delle specifiche di test, gli script generati dalle specifiche di test e i log dello script delle specifiche di test vengono tutti forniti come elementi scaricabili al termine dell'esecuzione del test. Ciò può portare alla divulgazione involontaria di segreti per gli altri utenti del tuo account con accesso in lettura all'esecuzione del test.

Migrazione dei test da un ambiente di test standard a un ambiente di test personalizzato

Puoi passare da una modalità di esecuzione di test standard a una modalità di esecuzione personalizzata in AWS Device Farm. La migrazione prevede principalmente due diverse forme di esecuzione:

1. Modalità standard: questa modalità di esecuzione dei test è progettata principalmente per fornire ai clienti report granulari e un ambiente completamente gestito.
2. Modalità personalizzata: questa modalità di esecuzione dei test è progettata per diversi casi d'uso che richiedono esecuzioni di test più rapide, la possibilità di passare da un modello all'altro e raggiungere la parità con l'ambiente locale e lo streaming video in diretta.

Per ulteriori informazioni sulle modalità standard e personalizzate in Device Farm, vedere Ambienti di test in AWS Device Farm eAmbienti di test personalizzati in AWS Device Farm.

Considerazioni sulle tempistiche di migrazione

Questa sezione elenca alcuni dei principali casi d'uso da considerare durante la migrazione alla modalità personalizzata:

1. Velocità: nella modalità di esecuzione standard, Device Farm analizza i metadati dei test che hai impacchettato e caricato utilizzando le istruzioni di imballaggio per il tuo particolare framework. L'analisi rileva il numero di test nel pacchetto. Successivamente, Device Farm esegue ogni test separatamente e presenta i log, i video e gli altri artefatti dei risultati singolarmente per ogni test. Tuttavia, ciò aumenta costantemente il tempo totale di esecuzione dei end-to-end test, in quanto il servizio prevede la pre e post-elaborazione dei test e gli artefatti dei risultati.

   Al contrario, la modalità di esecuzione personalizzata non analizza il pacchetto di test; ciò significa nessuna preelaborazione e una post-elaborazione minima per i test o gli artefatti dei risultati. Ciò si traduce in tempi di end-to-end esecuzione totali vicini alla configurazione locale. I test vengono eseguiti nello stesso formato in cui lo sarebbero se fossero eseguiti sui computer locali. I risultati dei test sono gli stessi che si ottengono localmente e sono disponibili per il download al termine dell'esecuzione del lavoro.
2. Personalizzazione o flessibilità: la modalità di esecuzione standard analizza il pacchetto di test per rilevare il numero di test e quindi esegue ciascun test separatamente. Tieni presente che non è garantito che i test vengano eseguiti nell'ordine specificato. Di conseguenza, i test che richiedono una particolare sequenza di esecuzione potrebbero non funzionare come previsto. Inoltre, non è possibile personalizzare l'ambiente del computer host o trasmettere i file di configurazione che potrebbero essere necessari per eseguire i test in un determinato modo.

   Al contrario, la modalità personalizzata consente di configurare l'ambiente del computer host, inclusa la possibilità di installare software aggiuntivo, applicare filtri ai test, passare file di configurazione e controllare la configurazione di esecuzione dei test. Ciò avviene tramite un file yaml (chiamato anche file testspec) che è possibile modificare aggiungendovi comandi di shell. Questo file yaml viene convertito in uno script di shell che viene eseguito sulla macchina host di test. Puoi salvare più file yaml e sceglierne uno dinamicamente in base alle tue esigenze quando pianifichi un'esecuzione.
3. Video e registrazione in diretta: sia la modalità di esecuzione standard che quella personalizzata forniscono video e registri per i test. Tuttavia, in modalità standard, il video e i registri predefiniti dei test vengono visualizzati solo dopo il completamento dei test.

Al contrario, la modalità personalizzata offre un live streaming del video e dei registri lato client dei test. Inoltre, puoi scaricare il video e altri artefatti alla fine del/i test/i.

> **Tip**
>
> Se il tuo caso d'uso riguarda almeno uno dei fattori sopra indicati, ti consigliamo vivamente di passare alla modalità di esecuzione personalizzata.

Fasi della migrazione

Per migrare dalla modalità standard a quella personalizzata, procedi come segue:

1. Accedi Console di gestione AWS e apri la console Device Farm all'indirizzo https://console.aws.amazon.com/devicefarm/.
2. Scegli il tuo progetto e poi inizia una nuova esecuzione di automazione.
3. Carica la tua app (o `web app` selezionala), scegli il tipo di framework di test, carica il pacchetto di test, quindi, sotto il `Choose your execution environment` parametro, scegli l'opzione`Run your test in a custom environment`.
4. Per impostazione predefinita, il file delle specifiche di test di esempio di Device Farm viene visualizzato e modificato. Questo file di esempio può essere utilizzato come punto di partenza per provare i test in modalità ambiente personalizzato. Quindi, una volta verificato che i test funzionino correttamente dalla console, puoi modificare qualsiasi integrazione di API, CLI e pipeline con Device Farm per utilizzare questo file delle specifiche di test come parametro durante la pianificazione delle esecuzioni dei test. Per informazioni su come aggiungere un file delle specifiche di test come parametro per le esecuzioni, consulta la sezione relativa ai `testSpecArn` parametri per l'API nella nostra guida all'`ScheduleRun`API.

Framework Appium

In un ambiente di test personalizzato, Device Farm non inserisce né sostituisce alcuna funzionalità Appium nei test del framework Appium. Devi specificare le funzionalità di Appium del tuo test o nel file YAML di specifica del test o nel codice del test.

Strumentazione Android

Non devi apportare modifiche per spostare i tuoi test Instrumentation per Android in un ambiente di test personalizzato.

iOS XCUITest

Non è necessario apportare modifiche per spostare i XCUITest test iOS in un ambiente di test personalizzato.

Estensione degli ambienti di test personalizzati in Device Farm

AWS Device Farm consente di configurare un ambiente personalizzato per i test automatici (modalità personalizzata), che è l'approccio consigliato per tutti gli utenti di Device Farm. La modalità personalizzata Device Farm ti consente di eseguire più di una semplice suite di test. In questa sezione, imparerai come estendere la tua suite di test e ottimizzare i test.

Per ulteriori informazioni sugli ambienti di test personalizzati in Device Farm, vedereAmbienti di test personalizzati in AWS Device Farm.

Argomenti

- Impostazione di un PIN del dispositivo durante l'esecuzione dei test in Device Farm
- Accelerazione dei test basati su Appium in Device Farm grazie alle funzionalità desiderate
- Utilizzo di Webhook e altro APIs dopo l'esecuzione dei test in Device Farm
- Aggiungere file aggiuntivi al pacchetto di test in Device Farm

Impostazione di un PIN del dispositivo durante l'esecuzione dei test in Device Farm

Alcune applicazioni richiedono l'impostazione di un PIN sul dispositivo. Device Farm non supporta l'impostazione di un PIN sui dispositivi in modo nativo. Tuttavia, ciò è possibile con le seguenti avvertenze:

- Il dispositivo deve avere Android 8 o versioni successive.
- Il PIN deve essere rimosso al termine del test.

Per impostare il PIN nei test, utilizza le `post_test` fasi `pre_test` e per impostare e rimuovere il PIN, come illustrato di seguito:

```
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"
```

All'avvio della suite di test, viene impostato il PIN 1234. Dopo la chiusura della suite di test, il PIN viene rimosso.

> **Warning**
>
> Se non rimuovi il PIN dal dispositivo al termine del test, il dispositivo e il tuo account verranno messi in quarantena.

Per ulteriori modi per estendere la suite di test e ottimizzare i test, consultaEstensione degli ambienti di test personalizzati in Device Farm.

Accelerazione dei test basati su Appium in Device Farm grazie alle funzionalità desiderate

Quando usi Appium, potresti scoprire che la suite di test in modalità standard è molto lenta. Questo perché Device Farm applica le impostazioni predefinite e non fa ipotesi su come si desidera utilizzare l'ambiente Appium. Sebbene queste impostazioni predefinite siano basate sulle migliori pratiche del settore, potrebbero non essere applicabili alla vostra situazione. Per ottimizzare i parametri del server Appium, puoi regolare le funzionalità Appium predefinite nelle specifiche del test. Ad esempio, quanto segue imposta la `usePrebuildWDA` funzionalità `true` in una suite di test iOS per accelerare l'ora di avvio iniziale:

```
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 &
```

Le funzionalità di Appium devono essere una struttura JSON citata e con escape da shell.

Le seguenti funzionalità di Appium sono fonti comuni di miglioramento delle prestazioni:

`noReset` e `fullReset`

Queste due funzionalità, che si escludono a vicenda, descrivono il comportamento di Appium al termine di ogni sessione. Quando `noReset` è impostato su`true`, il server Appium non rimuove i dati dall'applicazione al termine di una sessione Appium, in effetti non esegue alcuna pulizia. `fullReset`disinstalla e cancella tutti i dati dell'applicazione dal dispositivo dopo la chiusura della sessione. Per ulteriori informazioni, consulta Reset Strategies nella documentazione di Appium.

`ignoreUnimportantViews`(Solo Android)

Indica ad Appium di comprimere la gerarchia dell'interfaccia utente Android solo nelle viste pertinenti per il test, velocizzando la ricerca di determinati elementi. Tuttavia, ciò può interrompere alcune suite di test XPath basate su di esse perché la gerarchia del layout dell'interfaccia utente è stata modificata.

`skipUnlock`(Solo Android)

Informa Appium che al momento non è impostato alcun codice PIN, il che velocizza i test dopo un evento di spegnimento dello schermo o un altro evento di blocco.

`webDriverAgentUrl`(solo iOS)

Indica ad Appium di presumere che una dipendenza iOS essenziale sia già in esecuzione e disponibile per accettare richieste HTTP all'URL specificato. `webDriverAgent` Se `webDriverAgent` non è già attivo e funzionante, Appium può impiegare del tempo all'inizio

di una suite di test per avviare. webDriverAgent Se lo avvii webDriverAgent da solo e lo fai http://localhost:8100 quando avvii Appium, puoi avviare la tua suite di test più velocemente. webDriverAgentUrl Nota che questa funzionalità non dovrebbe mai essere utilizzata insieme alla useNewWDA funzionalità.

È possibile utilizzare il codice seguente per iniziare webDriverAgent dal file delle specifiche di test sulla porta locale del dispositivo8100, quindi inoltrarlo alla porta locale dell'host di test 8100 (ciò consente webDriverAgentUrl di impostare il valore suhttp://localhost:8100). Questo codice deve essere eseguito durante la fase di installazione dopo aver definito qualsiasi codice per la configurazione di Appium e delle variabili di webDriverAgent ambiente:

```
      # 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 &
```

Quindi, puoi aggiungere il seguente codice al file delle specifiche di test per assicurarti che webDriverAgent sia avviato correttamente. Questo codice dovrebbe essere eseguito alla fine della fase di pre-test dopo aver verificato che Appium sia stato avviato correttamente:

```
      # 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;
```

Per ulteriori informazioni sulle funzionalità supportate da Appium, consulta Appium Desired Capabilities nella documentazione di Appium.

Per ulteriori modi per estendere la suite di test e ottimizzare i test, consulta. Estensione degli ambienti di test personalizzati in Device Farm

Utilizzo di Webhook e altro APIs dopo l'esecuzione dei test in Device Farm

È possibile fare in modo che Device Farm chiami un webhook al termine dell'utilizzo curl di ogni suite di test. Il processo per eseguire questa operazione varia a seconda della destinazione e della formattazione. Per il tuo webhook specifico, consulta la documentazione relativa a quel webhook. L'esempio seguente pubblica un messaggio ogni volta che una suite di test termina su un webhook Slack:

```
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
```

Per ulteriori informazioni sull'uso dei webhook con Slack, consulta Invio del primo messaggio Slack utilizzando Webhook nel riferimento all'API Slack.

Per ulteriori modi per estendere la suite di test e ottimizzare i test, consulta. Estensione degli ambienti di test personalizzati in Device Farm

Non sei limitato a usare per curl chiamare i webhook. I pacchetti di test possono includere script e strumenti aggiuntivi, purché siano compatibili con l'ambiente di esecuzione Device Farm. Ad

esempio, il pacchetto di test può includere script ausiliari che inoltrano richieste ad altri. APIs Assicurati che tutti i pacchetti richiesti siano installati insieme ai requisiti della tua suite di test. Per aggiungere uno script che viene eseguito dopo il completamento della suite di test, includi lo script nel pacchetto di test e aggiungi quanto segue alle specifiche di test:

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

> **Note**
>
> La manutenzione delle chiavi API o degli altri token di autenticazione utilizzati nel pacchetto di test è una tua responsabilità. Ti consigliamo di mantenere qualsiasi forma di credenziale di sicurezza al di fuori del controllo del codice sorgente, di utilizzare credenziali con il minor numero di privilegi possibile e di utilizzare token revocabili e di breve durata quando possibile. Per verificare i requisiti di sicurezza, consulta la documentazione della terza parte che utilizzi. APIs

Se prevedi di utilizzare AWS i servizi come parte della tua suite di esecuzione dei test, dovresti utilizzare le credenziali temporanee IAM, generate al di fuori della tua suite di test e incluse nel pacchetto di test. Queste credenziali devono avere il minor numero di autorizzazioni concesse e la durata di vita più breve possibile. Per ulteriori informazioni sulla creazione di credenziali temporanee, consulta Richiesta di credenziali di sicurezza temporanee nella Guida per l'utente IAM.

Per ulteriori modi per estendere la suite di test e ottimizzare i test, consulta. Estensione degli ambienti di test personalizzati in Device Farm

Aggiungere file aggiuntivi al pacchetto di test in Device Farm

Potresti voler utilizzare file aggiuntivi come parte dei tuoi test come file di configurazione aggiuntivi o dati di test aggiuntivi. Puoi aggiungere questi file aggiuntivi al tuo pacchetto di test prima di caricarlo AWS Device Farm, quindi accedervi dalla modalità ambiente personalizzata. Fondamentalmente, tutti i formati di caricamento dei pacchetti di test (ZIP, IPA, APK, JAR, ecc.) sono formati di archivio dei pacchetti che supportano le operazioni ZIP standard.

È possibile aggiungere file all'archivio di test prima di caricarlo AWS Device Farm utilizzando il seguente comando:

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

Per una directory di file aggiuntivi:

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

Questi comandi funzionano come previsto per tutti i formati di caricamento dei pacchetti di test ad eccezione dei file IPA. Per i file IPA, specialmente se utilizzati con XCUITests, si consiglia di inserire i file aggiuntivi in una posizione leggermente diversa a causa della progettazione dei pacchetti AWS Device Farm di test iOS. Quando crei il tuo test iOS, la directory dell'applicazione di test si troverà all'interno di un'altra directory denominata*`Payload`*.

Ad esempio, ecco come potrebbe apparire una di queste directory di test iOS:

```
$ 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
```

Per questi XCUITest pacchetti, aggiungete eventuali file aggiuntivi alla directory che termina all'*.app*interno della *Payload* directory. Ad esempio, i comandi seguenti mostrano come aggiungere un file a questo pacchetto di test:

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

Quando aggiungete un file al pacchetto di test, potete aspettarvi un comportamento di interazione leggermente diverso in AWS Device Farm base al formato di caricamento.
Se il caricamento ha utilizzato l'estensione del file ZIP, AWS Device Farm decomprimerà automaticamente il caricamento prima del test e lascerà i file decompressi nella posizione con la *$DEVICEFARM_TEST_PACKAGE_PATH* variabile di ambiente. (Ciò significa che se aggiungeste un file chiamato *extra_file* alla radice dell'archivio come nel primo esempio, esso si troverebbe in *$DEVICEFARM_TEST_PACKAGE_PATH/extra_file* durante il test).

Per usare un esempio più pratico, se sei un utente di Appium TestNG che desidera includere un *testng.xml* file nel test, puoi includerlo nel tuo archivio usando il seguente comando:

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

Quindi, puoi modificare il comando test nella modalità ambiente personalizzata nel modo seguente:

```
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
```

Se l'estensione per il caricamento del pacchetto di test non è ZIP (ad esempio, APK, IPA o JAR), il file del pacchetto caricato si trova in*$DEVICEFARM_TEST_PACKAGE_PATH*. Poiché si tratta ancora

di file in formato di archivio, puoi decomprimere il file per accedere ai file aggiuntivi dall'interno. Ad esempio, il comando seguente decomprimerà il contenuto del pacchetto di test (per file APK, IPA o JAR) nella directory: */tmp*

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

Nel caso di un file APK o JAR, i file aggiuntivi verranno decompressi */tmp* nella directory (ad esempio,). */tmp/extra_file* Nel caso di un file IPA, come spiegato in precedenza, i file aggiuntivi si troverebbero in una posizione leggermente diversa all'interno della cartella che termina con*.app*, che si trova all'interno della directory. *Payload* Ad esempio, in base all'esempio IPA precedente, il file si troverebbe nella posizione */tmp/Payload/ADFiOSReferenceAppUITests-Runner.app/extra_file* (referenziabile come). */tmp/Payload/*.app/extra_file*

Per ulteriori modi per estendere la suite di test e ottimizzare i test, consulta. Estensione degli ambienti di test personalizzati in Device Farm

Accesso remoto in AWS Device Farm

L'accesso remoto consente di eseguire gesti, operazioni di scorrimento e di interagire con un dispositivo tramite browser Web in tempo reale per testare la funzionalità e riprodurre i problemi dei clienti. Puoi interagire con un dispositivo specifico creando una sessione di accesso remoto con quel dispositivo.

Una sessione in Device Farm è un'interazione in tempo reale con un dispositivo fisico effettivo ospitato in un browser web. Una sessione mostra il singolo dispositivo selezionato all'avvio della sessione. Un utente può avviare più di una sessione alla volta con il numero totale di dispositivi simultanei tenendo conto del limite del numero di slot dei dispositivi di cui dispone. Puoi acquistare slot dei dispositivi in base alla famiglia di dispositivi (Android o iOS). Per ulteriori informazioni, consulta Prezzi di Device Farm.

Device Farm offre attualmente un sottoinsieme di dispositivi per il test di accesso remoto. Nuovi dispositivi vengono costantemente aggiunti al pool di dispositivi.

Device Farm acquisisce il video di ogni sessione di accesso remoto e genera registri delle attività durante la sessione. I risultati includono qualsiasi informazione che fornisci durante una sessione.

> **Note**
>
> Per motivi di sicurezza, ti consigliamo di non fornire o inserire informazioni sensibili come numeri di conti, login personale e altri dati durante una sessione di accesso remoto. Se possibile, utilizza alternative sviluppate specificamente per i test, come gli account di test.

Argomenti

Creazione di una sessione di accesso remoto in AWS Device Farm

Per ulteriori informazioni sulle sessioni di accesso remoto, consulta Sessioni.

Prerequisiti

- Crea un progetto in Device Farm. Segui le istruzioni riportate in Creazione di un progetto in AWS Device Farm, poi torna a questa pagina.

Crea una sessione remota

Console

1. Accedere alla console Device Farm all'indirizzo https://console.aws.amazon.com/devicefarm.
2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.
3. Se hai già un progetto, selezionalo dall'elenco. Altrimenti, crea un progetto seguendo le istruzioni riportate inCreazione di un progetto in AWS Device Farm.
4. Nella scheda Accesso remoto, scegli Crea sessione di accesso remoto.
5. Selezionare un dispositivo per la propria sessione. Puoi scegliere dall'elenco dei dispositivi disponibili o cercare un dispositivo utilizzando la barra di ricerca nella parte superiore dell'elenco.
6. (Facoltativo) Includi un'app e app ausiliarie come parte della sessione. Queste possono essere app caricate di recente o app caricate in precedenza in questo progetto negli ultimi 30 giorni (dopo 30 giorni, i caricamenti delle app scadranno).
7. In Session name (Nome sessione), immettere un nome per la sessione.
8. Selezionare Confirm and start session (Conferma e avvia sessione).

AWS CLI

Nota: queste istruzioni si concentrano solo sulla creazione di una sessione di accesso remoto. Per istruzioni su come caricare un'app da utilizzare durante la sessione, consulta Automatizzare i caricamenti delle app.

Innanzitutto, verifica che la tua versione CLI di AWS sia disponibile up-to-date scaricando e installando la versione più recente.

> **Important**
>
> Alcuni comandi menzionati in questo documento non sono disponibili nelle versioni precedenti dell'interfaccia a riga di comando di AWS.

Quindi, puoi determinare su quale dispositivo desideri eseguire il test:

```
$ aws devicefarm list-devices
```

Questo mostrerà un output come il seguente:

```
{
    "devices":
    [
        {
            "arn": "arn:aws:devicefarm:us-
west-2::device:DE5BD47FF3BD42C3A14BF7A6EFB1BFE7",
            "name": "Google Pixel 8",
            "remoteAccessEnabled": true,
            "availability": "HIGHLY_AVAILABLE"
            ...
        },
        ...
    ]
}
```

Quindi, puoi creare la tua sessione di accesso remoto con un ARN del dispositivo a tua scelta:

```
$ 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
      ]
  }'
```

Questo mostrerà un output come il seguente:

```
{
    "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",
        ...
}
```

Ora, facoltativamente, possiamo effettuare un sondaggio e attendere che la sessione sia pronta:

```
$ 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

Nota: queste istruzioni si concentrano solo sulla creazione di una sessione di accesso remoto. Per istruzioni su come caricare un'app da utilizzare durante la sessione, consulta Automatizzare i caricamenti delle app.

Questo esempio trova innanzitutto qualsiasi dispositivo Google Pixel disponibile su Device Farm, quindi crea una sessione di accesso remoto con esso e attende l'esecuzione della sessione.

```
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

Nota: queste istruzioni si concentrano solo sulla creazione di una sessione di accesso remoto. Per istruzioni su come caricare un'app da utilizzare durante la sessione, consulta Automatizzare i caricamenti delle app.

Nota: questo esempio utilizza l'SDK AWS per Java v2 ed è compatibile con le versioni JDK 11 e successive.

Questo esempio trova innanzitutto qualsiasi dispositivo Google Pixel disponibile su Device Farm, quindi crea una sessione di accesso remoto con esso e attende l'esecuzione della sessione.

```
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

Nota: queste istruzioni si concentrano solo sulla creazione di una sessione di accesso remoto. Per istruzioni su come caricare un'app da utilizzare durante la sessione, consulta Automatizzare i caricamenti delle app.

Nota: questo esempio utilizza AWS SDK per JavaScript v3.

Questo esempio trova innanzitutto qualsiasi dispositivo Google Pixel disponibile su Device Farm, quindi crea una sessione di accesso remoto con esso e attende l'esecuzione della sessione.

```
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#

Nota: queste istruzioni si concentrano solo sulla creazione di una sessione di accesso remoto. Per istruzioni su come caricare un'app da utilizzare durante la sessione, consulta Automatizzare i caricamenti delle app.

Questo esempio trova innanzitutto qualsiasi dispositivo Google Pixel disponibile su Device Farm, quindi crea una sessione di accesso remoto con esso e attende l'esecuzione della sessione.

```
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

Nota: queste istruzioni si concentrano solo sulla creazione di una sessione di accesso remoto. Per istruzioni su come caricare un'app da utilizzare durante la sessione, consulta Automatizzare i caricamenti delle app.

Questo esempio trova innanzitutto qualsiasi dispositivo Google Pixel disponibile su Device Farm, quindi crea una sessione di accesso remoto con esso e attende l'esecuzione della sessione.

```
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
```

Fasi successive

Device Farm avvia la sessione non appena il dispositivo e l'infrastruttura richiesti sono disponibili, in genere entro pochi minuti. La finestra di dialogo Device Requested viene visualizzata fino all'avvio della sessione. Per annullare la richiesta di sessione, selezionare Cancel request (Annulla richiesta).

Se il dispositivo selezionato non è disponibile o occupato, lo stato della sessione viene visualizzato come Dispositivo in sospeso, a indicare che potrebbe essere necessario attendere qualche istante prima che il dispositivo sia disponibile per il test.

Se il tuo account ha raggiunto il limite di contemporaneità per i dispositivi pubblici con o senza contabilizzazione, lo stato della sessione viene visualizzato come In sospeso. Per quanto riguarda gli slot per dispositivi illimitati, puoi aumentare la concorrenza acquistando più slot per dispositivi. Per pay-as-you-go i dispositivi a consumo, contatta AWS tramite un ticket di supporto per richiedere un aumento della quota di servizio.

Quando inizia la configurazione della sessione, mostra prima lo stato In corso, quindi lo stato di Connessione mentre il browser Web locale tenta di aprire una connessione remota al dispositivo.

Dopo che una sessione viene avviata, se chiudi il browser o la scheda del browser senza arrestare la sessione o se la connessione tra il browser e Internet viene persa, la sessione rimarrà attiva per cinque minuti da quel momento. Dopodiché, Device Farm termina la sessione. Il tuo account verrà addebitato per i tempi di inattività.

Dopo l'inizio della sessione, puoi interagire con il dispositivo nel browser web o testare il dispositivo utilizzando Appium.

Utilizzo di una sessione di accesso remoto in AWS Device Farm

Per informazioni sull'esecuzione dei test interattivi delle app Android e iOS tramite le sessioni di accesso remoto, consulta Sessioni.

Prerequisiti

- Crea una sessione. Segui le istruzioni riportate in Creazione di una sessione, poi torna a questa pagina.

Utilizzare una sessione nella console Device Farm

Quando il dispositivo che hai richiesto per una sessione di accesso remoto diventa disponibile, la console mostra lo schermo del dispositivo. La sessione ha una durata massima di 150 minuti. Il tempo rimanente nella sessione viene visualizzato nel campo Tempo rimasto accanto al nome del dispositivo.

Installazione di un'applicazione

Per installare un'applicazione sul dispositivo di sessione, in Installa applicazioni, seleziona Scegli file, quindi scegli il file.apk (Android) o il file.ipa (iOS) che desideri installare. Le applicazioni che esegui in una sessione di accesso remoto non richiedono alcuna strumentazione di test o provisioning.

> **Note**
>
> Quando carichi un'app, a volte potrebbe essere disponibile con un po' di ritardo. Apparirà un messaggio di conferma per informarti se l'app è stata installata correttamente o meno.

Controllo del dispositivo

Puoi interagire in remoto con il dispositivo visualizzato nella console con le stesse modalità del dispositivo fisico, usando il mouse o altro dispositivo touch e la tastiera su schermo. Per i dispositivi Android, sono presenti pulsanti in View controls (Visualizza controlli) che hanno la stessa funzione dei pulsanti Home e Back (Indietro) di un dispositivo Android. Per i dispositivi iOS, è presente un pulsante Home che ha la stessa funzione del pulsante Home su un dispositivo iOS. Puoi anche passare da un'applicazione all'altra in esecuzione sul dispositivo scegliendo App recenti.

Passaggio dalla modalità verticale a quella orizzontale

Puoi anche passare dalla modalità ritratto (verticale) a quella orizzontale (orizzontale) per i dispositivi che stai utilizzando.

Fasi successive

Device Farm continua la sessione finché non viene interrotta manualmente o non viene raggiunto il limite di 150 minuti. Per terminare la sessione, scegli Arresta sessione. Una volta interrotta la sessione puoi accedere al video acquisito e ai log generati. Per ulteriori informazioni, consulta Recupero dei risultati della sessione.

Suggerimenti e trucchi

In alcune AWS regioni potrebbero verificarsi problemi di prestazioni con la sessione di accesso remoto. Ciò è dovuto in parte alla latenza che si verifica in alcune regioni. In caso di problemi a livello di prestazioni, concedi alla sessione di accesso remoto il tempo di recuperare prima di interagire nuovamente con l'app.

Recupero dei risultati di una sessione di accesso remoto in AWS Device Farm

Per informazioni sulle sessioni, consulta Sessioni.

- Prerequisiti
- Visualizzazione dei dettagli della sessione
- Scaricamento del video o dei registri della sessione

Prerequisiti

- Completa una sessione. Segui le istruzioni riportate in Utilizzo di una sessione di accesso remoto in AWS Device Farm, poi torna a questa pagina.

Visualizzazione dei dettagli della sessione

Al termine di una sessione di accesso remoto, la console Device Farm visualizza una tabella che contiene dettagli sull'attività durante la sessione. Per ulteriori informazioni, consulta Analisi delle informazioni di log.

Per tornare ai dettagli di una sessione in un secondo momento:

1. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.
2. Scegliete il progetto contenente la sessione.
3. Scegli Accesso remoto, quindi scegli la sessione che desideri rivedere dall'elenco.

Scaricamento del video o dei registri della sessione

Al termine di una sessione di accesso remoto, la console Device Farm fornisce l'accesso a un'acquisizione video della sessione e dei registri delle attività. Nei risultati della sessione, selezionare la scheda Files (File) per un elenco di collegamenti ai log e video della sessione. È possibile visualizzare questi file all'interno del browser o salvarli in locale.

Test di Appium in AWS Device Farm

Durante una sessione di accesso remoto, puoi eseguire test Appium dal tuo ambiente locale, indirizzando il dispositivo della sessione utilizzando un endpoint Appium gestito. Con un endpoint Appium, sei in grado di sviluppare, testare ed eseguire codice Appium con feedback rapidi e iterazioni rapide. Questo approccio ai test lato client offre la flessibilità necessaria per connettersi a un dispositivo Device Farm da qualsiasi ambiente client Appium di vostra scelta.

A complemento dei test lato client, Device Farm supporta anche l'esecuzione di test sull'infrastruttura gestita dal servizio, denominati esecuzione lato server. Con questo approccio, puoi caricare l'app e i test sul servizio, quindi eseguire i test in parallelo su più dispositivi utilizzando host di test gestiti dal servizio. Questo approccio si adatta bene ai test su molti dispositivi in modo indipendente, nonché ai test nel contesto di una pipeline. CI/CD

Per ulteriori informazioni sull'esecuzione lato server, consulta. Framework di test e test integrati

Argomenti

Cos'è un endpoint Appium?

Appium è un popolare framework di test software open source per testare applicazioni Web native, ibride e mobili su diversi dispositivi, inclusi telefoni cellulari e tablet, sia per iOS che per Android. Consente agli sviluppatori e agli ingegneri del QA (Quality Assurance) di scrivere script in grado di controllare in remoto un dispositivo, simulare le interazioni degli utenti e verificare che l'applicazione in test si comporti come previsto. Appium interagisce con le app dal punto di vista dell'utente finale, consentendo ai tester di sviluppare test che simulano il modo in cui gli utenti reali utilizzeranno l'app per i loro test.

Appium è basato sul modello client-server, in cui un client locale richiede a un server Appium (locale o remoto) di comandare un dispositivo per proprio conto. Il server Appium gestisce un driver per la

comunicazione con il dispositivo, ad esempio il UIAutomator2 driver per Android o il driver XCUITest per iOS. Tutti i comandi seguono gli WebDriver standard W3C per il controllo di un dispositivo.

L'endpoint Appium di Device Farm espone un URL del server Appium per il dispositivo nella sessione di accesso remoto. L'URL dell'endpoint Appium sarà specifico per quel dispositivo in quella sessione e rimarrà valido per tutta la durata della sessione, consentendoti di eseguire iterazioni sullo stesso dispositivo senza tempi di configurazione aggiuntivi. Per ulteriori informazioni sull'accesso remoto, consulta. Accesso remoto

Guida introduttiva ai test di Appium

Per la maggior parte degli utenti di Appium, l'utilizzo di Device Farm per i test di Appium richiede solo piccole modifiche alla configurazione di test esistente.

Ad alto livello, ci sono tre passaggi per utilizzare Device Farm per i test Appium lato client:

1. Innanzitutto, è necessario creare una sessione di accesso remoto per testare un dispositivo Device Farm. Puoi includere le app come parte della richiesta di accesso remoto o installare le app dopo l'inizio della sessione.
2. Una volta che la sessione è in esecuzione, puoi copiare l'URL dell'endpoint Appium e utilizzarlo tramite uno strumento autonomo (come Appium Inspector) o dal codice di test Appium nel tuo IDE. L'URL sarà valido per tutta la durata della sessione di accesso remoto.
3. Infine, una volta avviato il test Appium, puoi rivedere i log del server Appium in tempo reale durante l'esecuzione del test insieme al flusso video del tuo dispositivo.

Interazione con il dispositivo tramite Appium

Dopo aver creato una sessione di accesso remoto, il dispositivo sarà disponibile per i test di Appium. Per l'intera durata della sessione di accesso remoto, puoi eseguire tutte le sessioni Appium che desideri sul dispositivo, senza limiti ai client che utilizzi. Ad esempio, puoi iniziare eseguendo un test utilizzando il codice Appium locale dal tuo IDE, quindi passare all'utilizzo di Appium Inspector per risolvere eventuali problemi riscontrati. La sessione può durare fino a 150 minuti, tuttavia, se non vi è alcuna attività per più di 5 minuti (tramite la console interattiva o tramite l'endpoint Appium), la sessione scadrà.

Utilizzo delle app per i test con la sessione Appium

Device Farm ti consente di utilizzare le tue app come parte della richiesta di creazione di una sessione di accesso remoto o di installare app durante la sessione di accesso remoto stessa. Queste app vengono installate automaticamente sul dispositivo in esame e vengono inserite come funzionalità predefinite per qualsiasi richiesta di sessione Appium. Quando crei una sessione di accesso remoto, hai la possibilità di passare un'app ARN, che verrà utilizzata di default come `appium:app` funzionalità per tutte le sessioni Appium successive, nonché un'app ausiliaria ARNs, che verrà utilizzata come funzionalità. `appium:otherApps`

Ad esempio, se crei una sessione di accesso remoto utilizzando un'app `com.aws.devicefarm.sample` come app e `com.aws.devicefarm.other.sample` come una delle tue app ausiliarie, quando vai a creare una sessione Appium, avrà funzionalità simili alle seguenti:

```
{
    "value":
    {
        "sessionId": "abcdef123456-1234-5678-abcd-abcdef123456",
        "capabilities":
        {
            "app": "/tmp/com.aws.devicefarm.sample.apk",
            "otherApps": "[\"/tmp/com.aws.devicefarm.other.sample.apk\"]",
            ...
        }
    }
}
```

Durante la sessione, puoi installare app aggiuntive (all'interno della console o utilizzando l'[InstallToRemoteAccessSession](#)API). Queste sostituiranno tutte le app esistenti utilizzate in precedenza come `appium:app` funzionalità. Se le app utilizzate in precedenza hanno un nome di pacchetto distinto, rimarranno sul dispositivo e verranno utilizzate come parte della `appium:otherApps` funzionalità.

Ad esempio, se inizialmente utilizzi un'app `com.aws.devicefarm.sample` durante la creazione della sessione di accesso remoto, ma poi installi una nuova app denominata `com.aws.devicefarm.other.sample` durante la sessione, le sessioni di Appium avranno funzionalità simili alle seguenti:

```
{
```

```
    "value":
    {
        "sessionId": "abcdef123456-1234-5678-abcd-abcdef123456",
        "capabilities":
        {
            "app": "/tmp/com.aws.devicefarm.other.sample.apk",
            "otherApps": "[\"/tmp/com.aws.devicefarm.sample.apk\"]",
            ...
        }
    }
}
```

Se preferisci, puoi specificare in modo esplicito le funzionalità della tua app utilizzando il nome dell'app (utilizzando le `appium:bundleId` funzionalità `appium:appPackage` o rispettivamente per Android e iOS).

Se stai testando un'app web, specifica la `browserName` funzionalità per la tua richiesta di creazione di sessioni Appium. Il `Chrome` browser è disponibile su tutti i dispositivi Android e il `Safari` browser è disponibile su tutti i dispositivi iOS.

Device Farm non supporta il passaggio di un URL remoto o di un percorso di file system locale `appium:app` durante una sessione di accesso remoto. Carica le app su Device Farm e includile invece nella sessione.

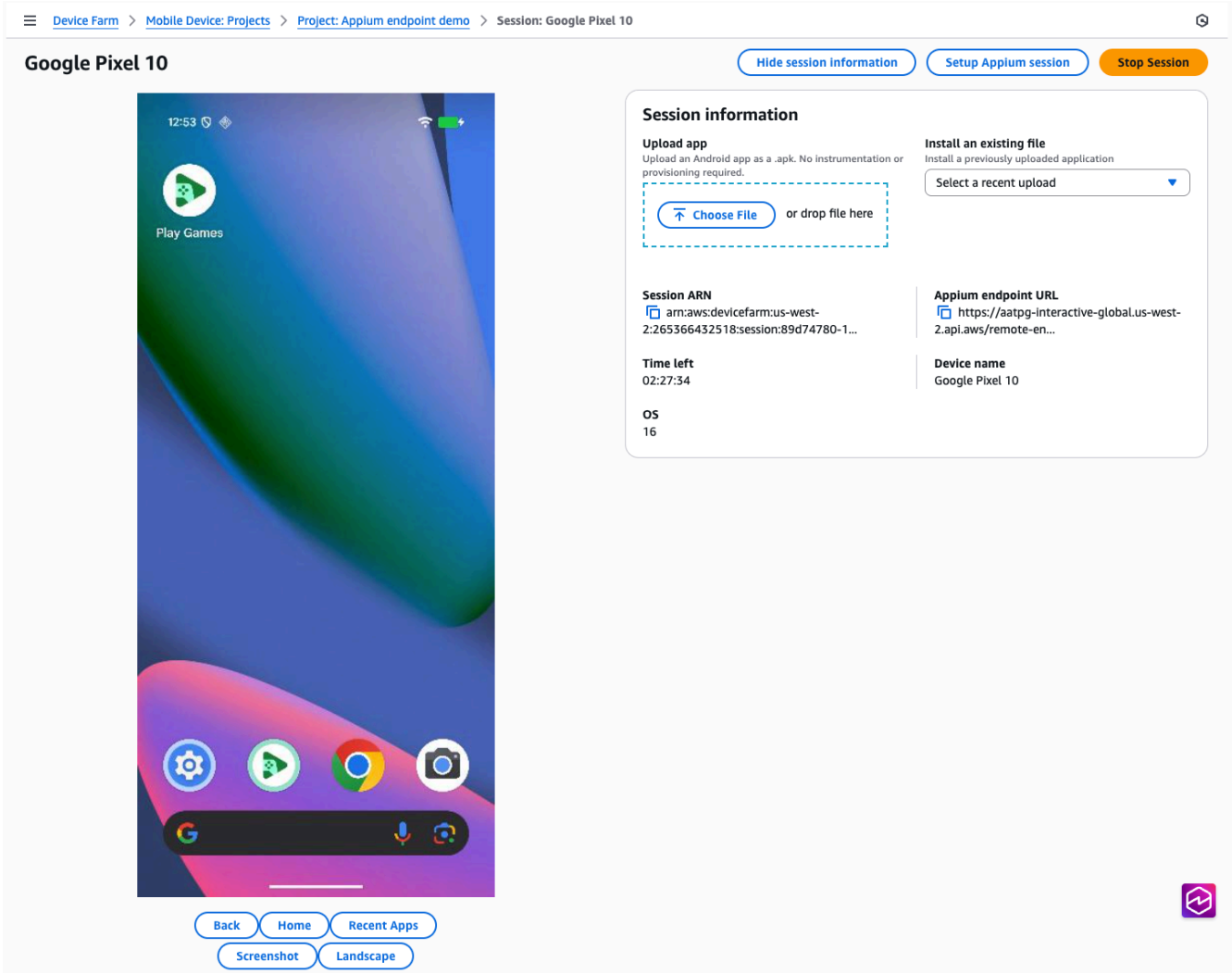
> **Note**
>
> Per ulteriori informazioni sul caricamento automatico delle app come parte della sessione di accesso remoto, consulta la sezione Automatizzazione dei caricamenti delle app.

Come usare l'endpoint Appium

Ecco i passaggi per accedere all'endpoint Appium della sessione dalla console, dal e dal. AWS CLI AWS SDKs Questi passaggi includono come iniziare a eseguire i test utilizzando vari framework di test dei client Appium:

Console

1. Apri la pagina della sessione di accesso remoto nel tuo browser web:



2. Per eseguire una sessione utilizzando Appium Inspector, procedi come segue:

   a. Fai clic sul pulsante Setup Appium session

   b. Segui le istruzioni sulla pagina per iniziare una sessione utilizzando Appium Inspector.

3. Per eseguire un test Appium dal tuo IDE locale, procedi come segue:

   a. Fai clic sull'icona «copia» accanto al testo Appium endpoint URL

   b. Incolla questo URL nel codice Appium locale ovunque tu specifichi attualmente il tuo indirizzo remoto o l'esecutore di comandi. Per esempi specifici della lingua, fai clic su una delle schede in questa finestra di esempio per la lingua che preferisci.

AWS CLI

Innanzitutto, verifica che la tua versione CLI di AWS sia disponibile up-to-date scaricando e installando la versione più recente.

> **Important**
>
> Il campo endpoint Appium non è disponibile nelle versioni precedenti dell'interfaccia a riga di comando di AWS.

Una volta che la sessione sarà attiva e funzionante, l'URL dell'endpoint Appium sarà disponibile tramite un campo denominato `remoteDriverEndpoint` nella risposta a una chiamata all'API: `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"
```

Questo mostrerà un output come il seguente:

```
{
    "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...",
        ...
}
```

È possibile utilizzare questo URL nel codice Appium locale ovunque si specifichi attualmente l'indirizzo remoto o l'esecutore di comandi. Per esempi specifici della lingua, fai clic su una delle schede in questa finestra di esempio per la lingua che preferisci.

Per un esempio di come interagire con l'endpoint direttamente dalla riga di comando, puoi usare lo strumento da riga di comando curl per chiamare direttamente un endpoint: WebDriver

```
$ curl "https://devicefarm-interactive-global.us-west-2.api.aws/remote-endpoint/
ABCD1234.../status"
```

Questo mostrerà un output come il seguente:

```
{
    "value":
    {
        "ready": true,
        "message": "The server is ready to accept new connections",
        "build":
        {
            "version": "2.5.1"
        }
    }
}
```

Python

Una volta che la sessione sarà attiva e funzionante, l'URL dell'endpoint Appium sarà disponibile tramite un campo denominato `remoteDriverEndpoint` nella risposta a una chiamata all'API: GetRemoteAccessSession

```
# 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

Nota: questo esempio utilizza l' AWS SDK for Java v2 ed è compatibile con le versioni JDK 11 e successive.

Una volta che la sessione sarà attiva e funzionante, l'URL dell'endpoint Appium sarà disponibile tramite un campo denominato `remoteDriverEndpoint` nella risposta a una chiamata all'API: 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

Nota: questo esempio utilizza AWS SDK per JavaScript v3 e WebDriveRio v8+ utilizzando Node 18+.

Una volta che la sessione sarà attiva e funzionante, l'URL dell'endpoint Appium sarà disponibile tramite un campo denominato nella risposta a una chiamata `remoteDriverEndpoint` all'API: [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#

Una volta che la sessione sarà attiva e funzionante, l'URL dell'endpoint Appium sarà disponibile tramite un campo denominato `remoteDriverEndpoint` nella risposta a una chiamata all'API: [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

Una volta che la sessione sarà attiva e funzionante, l'URL dell'endpoint Appium sarà disponibile tramite un campo denominato `remoteDriverEndpoint` nella risposta a una chiamata all'API: `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)
```

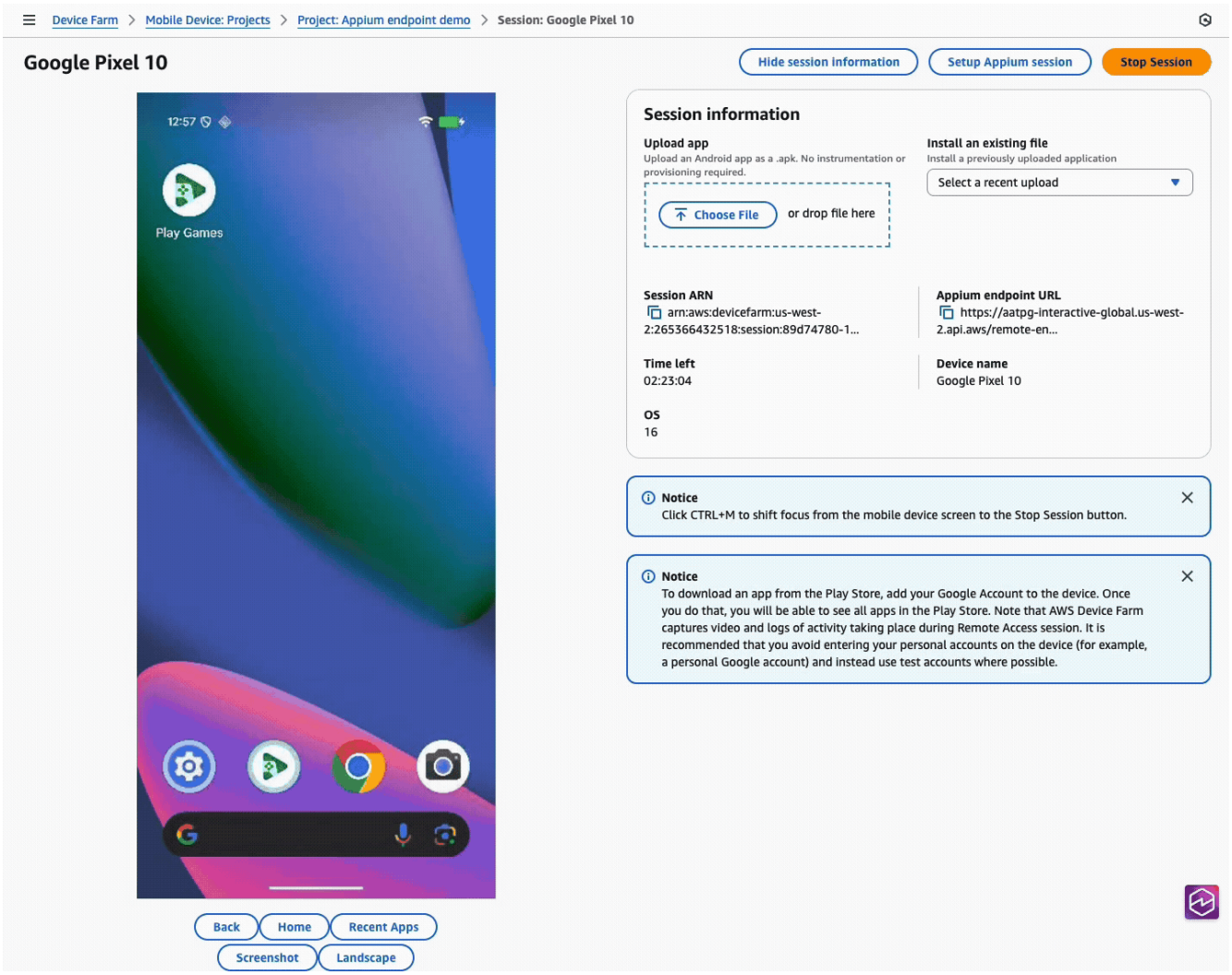
```
  endpoint = resp.remote_access_session.endpoints.remote_driver_endpoint
  raise "remote_driver_endpoint missing from response" if endpoint.nil? ||
 endpoint.empty?
  endpoint
end

# 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
```

Revisione dei log del server Appium

Dopo aver avviato una sessione Appium, puoi visualizzare i log del server Appium in tempo reale nella console Device Farm o scaricarli al termine della sessione di accesso remoto. Ecco le istruzioni per farlo:

Console

1. Nella console Device Farm, apri la sessione di accesso remoto per il tuo dispositivo.
2. Avvia una sessione di endpoint Appium con il dispositivo dall'IDE locale o da Appium Inspector
3. Quindi, il registro del server Appium verrà visualizzato accanto al dispositivo nella pagina della sessione di accesso remoto, con le «informazioni sulla sessione» disponibili nella parte inferiore della pagina sotto il dispositivo:



AWS CLI

Nota: questo esempio utilizza lo strumento da riga di comando *curl* per estrarre il registro da Device Farm.

Durante o dopo la sessione, puoi utilizzare l'ListArtifactsAPI di Device Farm per scaricare il log del server Appium.

```
$ 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
```

Questo mostrerà un output come il seguente durante la sessione:

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

E quanto segue al termine della sessione:

```
{
    "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..."
```

Questo mostrerà un output come il seguente:

```
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

Nota: questo esempio utilizza il *`requests`* pacchetto di terze parti per scaricare il registro, oltre all' AWS SDK per Python*`boto3`*.

Durante o dopo la sessione, puoi utilizzare l'`ListArtifacts`API di Device Farm per recuperare l'URL del registro del server Appium, quindi scaricarlo.

```
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()
```

Questo mostrerà un output come il seguente:

```
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

Nota: questo esempio utilizza l' AWS SDK for Java v2 *`HttpClient`* e per scaricare il registro ed è compatibile con le versioni JDK 11 e successive.

Durante o dopo la sessione, puoi utilizzare l'ListArtifactsAPI di Device Farm per recuperare l'URL del registro del server Appium e scaricarlo.

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

Questo mostrerà un output come il seguente:

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', ..., useDrivers: [ 'uiautomator' ] }
```

JavaScript

Nota: questo esempio utilizza AWS SDK for JavaScript (v3) e Node 18+ *`fetch`* per scaricare il registro.

Durante o dopo la sessione, puoi utilizzare l'`ListArtifacts`API di Device Farm per recuperare l'URL del registro del server Appium e scaricarlo.

```
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);
```

Questo mostrerà un output come il seguente:

```
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#

Nota: questo esempio utilizza l' AWS SDK for .NET *`HttpClient`* e per scaricare il registro.

Durante o dopo la sessione, puoi utilizzare l'`ListArtifacts`API di Device Farm per recuperare l'URL del registro del server Appium e scaricarlo.

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

Questo mostrerà un output come il seguente:

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', ..., useDrivers: [ 'uiautomator' ] }
```

Ruby

Nota: questo esempio utilizza l' AWS SDK for *`Net::HTTP`* Ruby e per scaricare il registro.

Durante o dopo la sessione, puoi utilizzare l'ListArtifactsAPI di Device Farm per recuperare l'URL del registro del server Appium e scaricarlo.

```
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)}"
```

Questo mostrerà un output come il seguente:

```
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' ] }
```

Funzionalità e comandi Appium supportati

L'endpoint Appium di Device Farm supporta la maggior parte degli stessi comandi e delle funzionalità desiderate che usi sui dispositivi locali, con alcune eccezioni. Gli elenchi seguenti mostrano quali funzionalità e comandi non sono attualmente supportati. Se i test non riescono a essere eseguiti come previsto a causa di una funzionalità limitata, apri una richiesta di assistenza per ulteriori informazioni.

Funzionalità supportate

Quando crei una sessione Appium su Device Farm, ti consigliamo di disporre di un set distinto di funzionalità che escluda qualsiasi funzionalità specifica del tuo dispositivo locale. In Device Farm, la creazione della sessione potrebbe non riuscire se sono impostate alcune funzionalità non supportate. Ciò include funzionalità specifiche del dispositivo come e. `udid` `platformVersion` Inoltre, alcune funzionalità ChromeDriver relative ad Android e WebDriverAgent iOS non sono supportate, così come le funzionalità supportate solo su emulatori e simulatori.

Comandi supportati

La maggior parte dei comandi Appium che vengono eseguiti correttamente su dispositivi Android e iOS reali verrà eseguita come previsto su Device Farm, con le seguenti esclusioni:

Comandi del dispositivo Appium () **`/appium/device`**

- `install_app`
- `finger_print`

- send_sms
- gsm_call
- gsm_signal
- gsm_voice
- power_ac
- power_capacity
- network_speed
- shake

Metodi e script di esecuzione di Appium () **/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`

Dispositivi privati in AWS Device Farm

Un dispositivo privato è un dispositivo mobile fisico che AWS Device Farm distribuisce per tuo conto in un data center Amazon. Questo dispositivo è esclusivo per il tuo AWS account.

> **Note**
>
> Attualmente, i dispositivi privati sono disponibili solo nella regione AWS Stati Uniti occidentali (Oregon) (`us-west-2`).

Se possiedi un parco istanze di dispositivi privati, è possibile creare sessioni di accesso remoto e pianificare sessioni di test tramite i dispositivi privati. Per ulteriori informazioni, consulta Creazione di un'esecuzione di test o avvio di una sessione di accesso remoto in AWS Device Farm. È anche possibile creare profili di istanza per controllare il comportamento dei dispositivi privati durante una sessione di accesso remoto o di test. Per ulteriori informazioni, consulta Creazione di un profilo di istanza in AWS Device Farm. Facoltativamente, puoi richiedere che determinati dispositivi privati Android vengano distribuiti come dispositivi rootati.

Puoi anche creare un servizio endpoint Amazon Virtual Private Cloud per testare app private a cui la tua azienda ha accesso ma che non sono raggiungibili tramite Internet. Ad esempio, potresti avere un'applicazione Web in esecuzione sulla tua VPC che desideri testare su dispositivi mobili. Per ulteriori informazioni, consulta Utilizzo dei servizi endpoint Amazon VPC con Device Farm - Legacy (non consigliato).

Se sei interessato a utilizzare una flotta di dispositivi privati, contattaci. Il team di Device Farm deve collaborare con te per configurare e distribuire una flotta di dispositivi privati per il tuo AWS account.

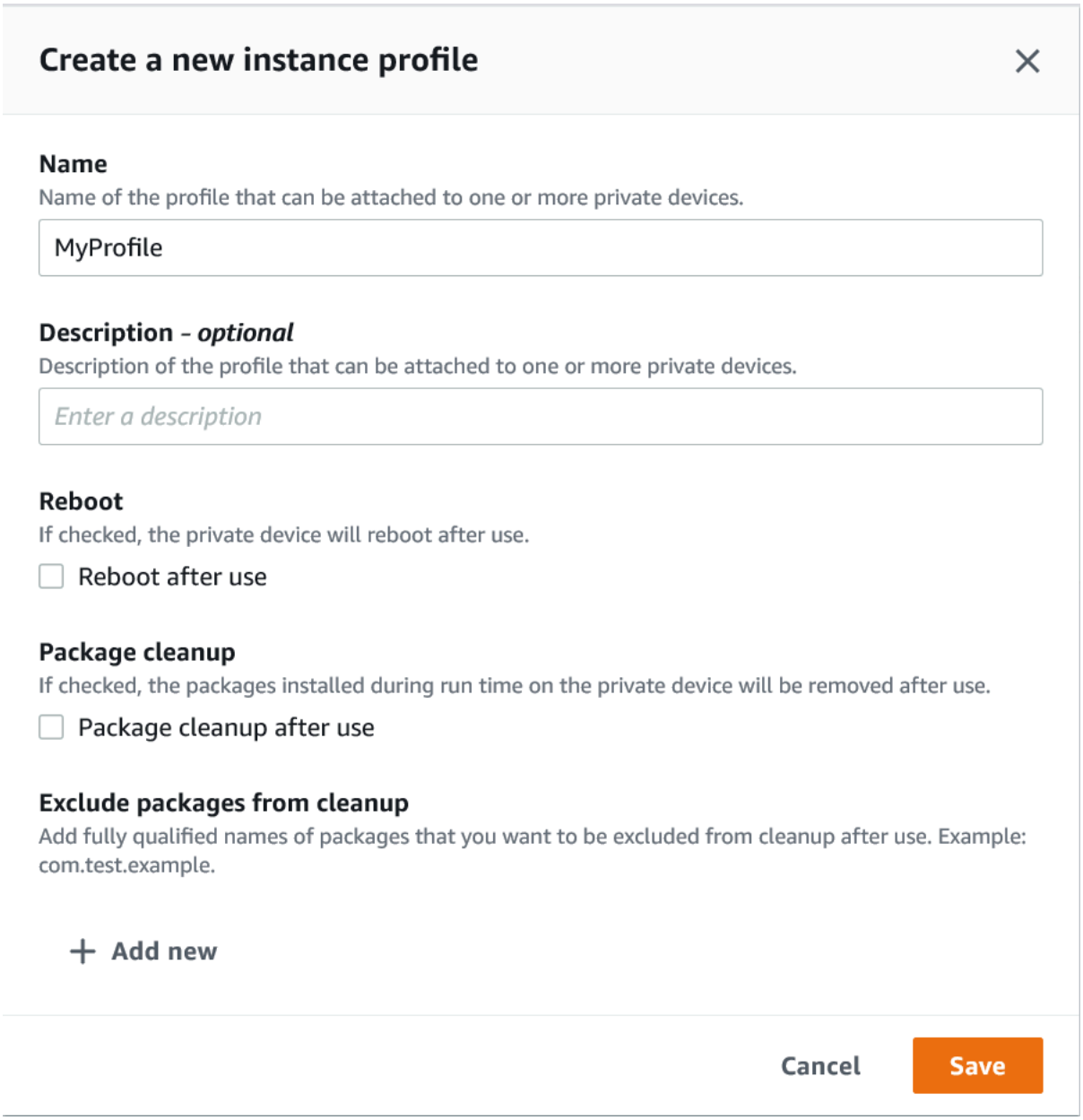
Argomenti

Creazione di un profilo di istanza in AWS Device Farm

È possibile impostare un parco istanze che contiene uno o più dispositivi privati. Questi dispositivi sono dedicati al tuo account AWS . Dopo aver configurato i dispositivi, è possibile creare uno o più profili di istanza per loro. I profili di istanza consentono di automatizzare le sessioni di test e di applicare le stesse impostazioni in maniera coerente alle istanze del dispositivo. I profili delle istanze possono anche aiutarti a controllare il comportamento della sessione di accesso remoto. Per ulteriori informazioni sui dispositivi privati in Device Farm, vedereDispositivi privati in AWS Device Farm.

Per creare un'istanza

1. Apri la console Device Farm all'indirizzo https://console.aws.amazon.com/devicefarm/.
2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Dispositivi privati.
3. Scegliere Instance profiles (Profili delle istanze).
4. Seleziona Crea profilo dell'istanza.
5. Immettere un nome per il profilo dell’istanza.



6. (Facoltativo) Inserire una descrizione per il profilo dell'istanza.
7. (Facoltativo) Modificate una delle seguenti impostazioni per specificare quali azioni desiderate che Device Farm esegua su un dispositivo al termine di ogni esecuzione di test o sessione:

   - Riavvia dopo l'uso: per riavviare il dispositivo, seleziona questa casella di controllo. Come impostazione predefinita, questa casella di controllo è deselezionata (`false`).
   - Pulizia dei pacchetti: per rimuovere tutti i pacchetti di app installati sul dispositivo, seleziona questa casella di controllo. Come impostazione predefinita, questa casella di controllo è deselezionata (`false`). Per mantenere tutte le app pacchetti installate sul dispositivo, deselezionare questa casella di controllo.

- Escludi pacchetti dalla pulizia: per mantenere solo i pacchetti di app selezionati sul dispositivo, seleziona la casella di controllo Package Cleanup, quindi scegli Aggiungi nuovo. Per il nome del pacchetto, inserire il nome completo del pacchetto app che si desidera conservare sul dispositivo (ad esempio `com.test.example`). Per mantenere più pacchetti app sul dispositivo, scegliere Add new (Aggiungi nuovo) e immettere il nome completo di ogni pacchetto.

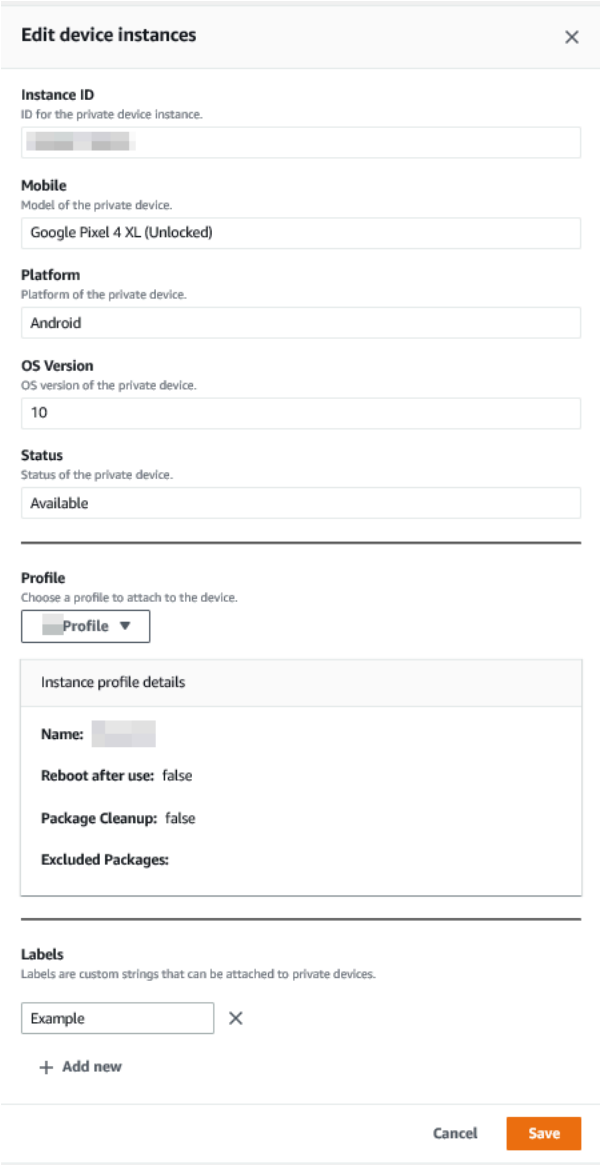
8. Scegli Save (Salva).

Richiedi dispositivi privati aggiuntivi in AWS Device Farm

In AWS Device Farm, puoi richiedere l'aggiunta di ulteriori istanze di dispositivi privati alla tua flotta. Puoi anche visualizzare e modificare le impostazioni delle istanze di dispositivi privati esistenti nella tua flotta. Per ulteriori informazioni sui dispositivi privati, consultaDispositivi privati in AWS Device Farm.

Per richiedere dispositivi privati aggiuntivi o modificarne le impostazioni

1. Apri la console Device Farm all'indirizzo https://console.aws.amazon.com/devicefarm/.
2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Dispositivi privati.
3. Scegliere Device instances (Istanze dispositivo). La scheda Device instances (Istanze dispositivo) visualizza una tabella dei dispositivi private del parco istanze. Per cercare o filtrare rapidamente la tabella, inserisci i termini di ricerca nella barra di ricerca sopra le colonne.
4. Per richiedere una nuova istanza di dispositivo privato, scegli Richiedi istanza di dispositivo o contattaci. I dispositivi privati richiedono una configurazione aggiuntiva con l'aiuto del team di Device Farm.
5. Nella tabella delle istanze del dispositivo, scegli l'opzione di attivazione/disattivazione accanto all'istanza su cui desideri visualizzare o gestire le informazioni, quindi scegli Modifica.



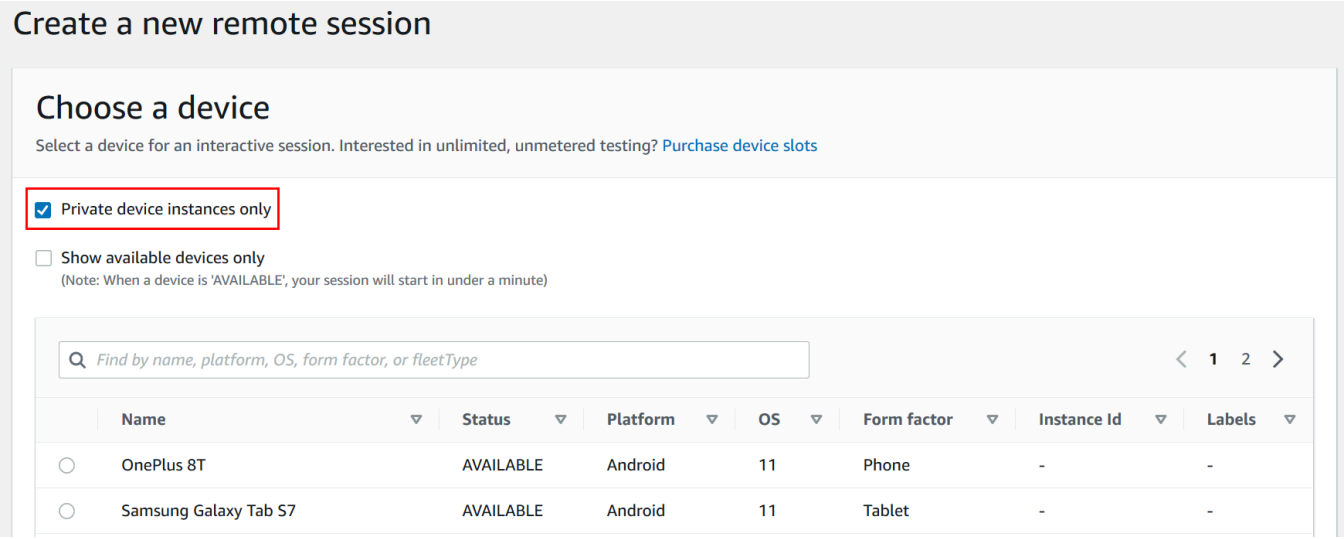
6. Per allegare un profilo di istanza all'istanza del dispositivo, selezionalo dall'elenco a discesa Profilo. Ad esempio, allegare un profilo di istanza può essere utile se desideri escludere sempre un pacchetto di app specifico dalle attività di pulizia. Per ulteriori informazioni sull'utilizzo dei profili di istanza con i dispositivi, consulta. Creazione di un profilo di istanza in AWS Device Farm
7. (Facoltativo) In Labels (Etichette), scegliere Add new (Aggiungi nuova) per aggiungere un'etichetta all'istanza del dispositivo. Le etichette possono aiutare a categorizzare i dispositivi e a trovare più facilmente i dispositivi specifici.
8. Scegli Save (Salva).

Creazione di un'esecuzione di test o avvio di una sessione di accesso remoto in AWS Device Farm

In AWS Device Farm, dopo aver configurato una flotta di dispositivi privati, puoi creare esecuzioni di test o avviare sessioni di accesso remoto con uno o più dispositivi privati della tua flotta. Per ulteriori informazioni sui dispositivi privati, consultaDispositivi privati in AWS Device Farm.

Per creare un test, eseguire o avviare una sessione di accesso remoto

1. Apri la console Device Farm all'indirizzo https://console.aws.amazon.com/devicefarm/.
2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.
3. Scegli un progetto esistente dall'elenco o creane uno nuovo. Per creare un nuovo progetto, scegliete Nuovo progetto, inserite un nome per il progetto, quindi scegliete Invia.
4. Esegui una delle seguenti operazioni:
   - Per creare una sessione di test, scegliere Automated tests (Test automatici) e Create a new run (Crea una nuova sessione). La procedura guidata fornisce delle istruzioni per la creazione della sessione. Per il passaggio Seleziona dispositivi, puoi modificare un pool di dispositivi esistente o creare un nuovo pool di dispositivi che includa solo i dispositivi privati che il team di Device Farm ha configurato e associato al tuo AWS account. Per ulteriori informazioni, consulta the section called “Crea un pool di dispositivi privato”.
   - Per avviare una sessione di accesso remoto, scegliere Remote access (Accesso remoto) e Start a new session (Avvia una nuova sessione). Nella pagina Scegli un dispositivo, seleziona Solo istanze di dispositivi privati per limitare l'elenco ai soli dispositivi privati che il team di Device Farm ha configurato e associato al tuo AWS account. Scegliere quindi il dispositivo al quale si desidera accedere, inserire un nome per la sessione di accesso remoto e scegliere Confirm and start session (Conferma e avvia sessione).



Selezione di dispositivi privati in un pool di dispositivi in AWS Device Farm

Per utilizzare dispositivi privati durante il test, puoi creare un pool di dispositivi che seleziona i tuoi dispositivi privati. I pool di dispositivi consentono di selezionare dispositivi privati principalmente attraverso tre tipi di regole del pool di dispositivi:

1. Regole basate sull'ARN del dispositivo
2. Regole basate sull'etichetta dell'istanza del dispositivo
3. Regole basate sull'ARN dell'istanza del dispositivo

Nelle sezioni seguenti, ogni tipo di regola e i relativi casi d'uso sono descritti in modo approfondito. È possibile utilizzare la console Device Farm, l'interfaccia a riga di AWS comando (AWS CLI) o l'API Device Farm per creare o modificare un pool di dispositivi con dispositivi privati utilizzando queste regole.

Argomenti

- Creazione di un pool di dispositivi privati con dispositivi privati (API)

ARN del dispositivo

L'ARN di un dispositivo è un identificatore che rappresenta un tipo di dispositivo anziché un'istanza fisica specifica del dispositivo. Un tipo di dispositivo è definito dai seguenti attributi:

- L'ID della flotta del dispositivo
- L'OEM del dispositivo
- Il numero di modello del dispositivo
- La versione del sistema operativo del dispositivo
- Lo stato del dispositivo che indica se è rootato o meno

Molte istanze di dispositivo fisico possono essere rappresentate da un singolo tipo di dispositivo in cui ogni istanza di quel tipo ha gli stessi valori per questi attributi. Ad esempio, se hai tre `Apple iPhone 13` dispositivi con versione iOS `16.1.0` nella tua flotta privata, ogni dispositivo condividerebbe lo stesso ARN del dispositivo. Se dal tuo parco dispositivi venissero aggiunti o rimossi gli stessi attributi, l'ARN del dispositivo continuerebbe a rappresentare tutti i dispositivi disponibili nel tuo parco dispositivi per quel tipo di dispositivo.

L'ARN dei dispositivi è il modo più efficace per selezionare i dispositivi privati per un pool di dispositivi perché consente al pool di dispositivi di continuare a selezionare i dispositivi indipendentemente dalle istanze specifiche del dispositivo che hai distribuito in un dato momento. Le singole istanze di dispositivi privati possono subire guasti hardware, pertanto Device Farm è tenuta a sostituirle automaticamente con nuove istanze funzionanti dello stesso tipo di dispositivo. In questi scenari, la regola ARN del dispositivo garantisce che il pool di dispositivi possa continuare a selezionare i dispositivi in caso di guasto hardware.

Quando utilizzi una regola ARN del dispositivo per i dispositivi privati nel tuo pool di dispositivi e pianifichi un'esecuzione di test con quel pool, Device Farm verificherà automaticamente quali istanze di dispositivo privato sono rappresentate dall'ARN di quel dispositivo. Tra le istanze attualmente disponibili, una di esse verrà assegnata per eseguire il test. Se al momento non sono disponibili istanze, Device Farm attenderà che la prima istanza disponibile dell'ARN di quel dispositivo diventi disponibile e la assegnerà per eseguire il test.

Etichette delle istanze del dispositivo

L'etichetta di un'istanza di dispositivo è un identificatore testuale che è possibile allegare come metadati per un'istanza di dispositivo. È possibile allegare più etichette a ciascuna istanza del dispositivo e la stessa etichetta a più istanze del dispositivo. Per ulteriori informazioni sull'aggiunta, la modifica o la rimozione delle etichette dei dispositivi dalle istanze del dispositivo, consulta Gestione dei dispositivi privati.

L'etichetta dell'istanza del dispositivo può essere un modo efficace per selezionare i dispositivi privati per un pool di dispositivi perché, se si dispone di più istanze di dispositivo con la stessa etichetta, consente al pool di dispositivi di selezionare una di esse per il test. Se l'ARN del dispositivo non è una buona regola per il tuo caso d'uso (ad esempio, se desideri scegliere tra dispositivi di più tipi di dispositivi o se desideri selezionare da un sottoinsieme di tutti i dispositivi di un tipo di dispositivo), le etichette delle istanze del dispositivo possono consentirti di scegliere tra più dispositivi per il tuo pool di dispositivi con maggiore granularità. Le singole istanze di dispositivi privati possono subire guasti hardware, pertanto Device Farm è tenuta a sostituirle automaticamente con nuove istanze funzionanti dello stesso tipo di dispositivo. In questi scenari, l'istanza del dispositivo sostitutivo non conserverà alcun metadato dell'etichetta di istanza del dispositivo sostituito. Pertanto, se applichi la stessa etichetta di istanza di dispositivo a più istanze di dispositivo, la regola di etichettatura delle istanze del dispositivo garantisce che il pool di dispositivi possa continuare a selezionare le istanze del dispositivo in caso di guasto hardware.

Quando utilizzi una regola di etichettatura delle istanze di dispositivo per i dispositivi privati nel tuo pool di dispositivi e pianifichi un'esecuzione di test con quel pool, Device Farm verificherà automaticamente quali istanze di dispositivo privato sono rappresentate da quell'etichetta di istanza del dispositivo e, tra quelle istanze, ne seleziona casualmente una disponibile per eseguire il test. Se non ce ne sono disponibili, Device Farm selezionerà casualmente qualsiasi istanza del dispositivo con l'etichetta dell'istanza del dispositivo per eseguire il test e metterà in coda il test da eseguire sul dispositivo una volta che sarà disponibile.

ARN istanza

L'ARN di un'istanza di dispositivo è un identificatore che rappresenta un'istanza fisica di dispositivo bare metal distribuita in una flotta privata. Ad esempio, se aveste tre *iPhone 13* dispositivi con sistema operativo *15.0.0* nella vostra flotta privata, mentre ogni dispositivo condividesse lo stesso ARN del dispositivo, ogni dispositivo avrebbe anche il proprio ARN di istanza che rappresenta solo quell'istanza.

L'ARN dell'istanza del dispositivo è il modo meno affidabile per selezionare i dispositivi privati per un pool di dispositivi ed è consigliato solo se le etichette del dispositivo ARNs e dell'istanza del dispositivo non si adattano al tuo caso d'uso. ARNs Le istanze di dispositivo vengono spesso utilizzate come regole per i pool di dispositivi quando un'istanza di dispositivo specifica è configurata in modo unico e specifico come prerequisito per il test e se è necessario conoscere e verificare tale configurazione prima di eseguire il test su di essa. Le singole istanze di dispositivi privati possono subire guasti hardware, pertanto Device Farm è tenuta a sostituirle automaticamente con nuove istanze funzionanti dello stesso tipo di dispositivo. In questi scenari, l'istanza del dispositivo sostitutiva avrà un ARN dell'istanza del dispositivo diverso rispetto al dispositivo sostituito. Quindi, se ti affidi all'istanza del dispositivo ARNs per il tuo pool di dispositivi, dovrai modificare manualmente la definizione delle regole del pool di dispositivi dall'utilizzo del vecchio ARN all'utilizzo del nuovo ARN. Se è necessario preconfigurare manualmente il dispositivo per il test, questo può essere un flusso di lavoro efficace (rispetto al dispositivo). ARNs Per eseguire test su larga scala, si consiglia di provare ad adattare questi casi d'uso alle etichette delle istanze del dispositivo e, se possibile, di preconfigurare più istanze del dispositivo per il test.

Quando utilizzi una regola ARN di istanza di dispositivo per i dispositivi privati nel tuo pool di dispositivi e pianifichi un'esecuzione di test con quel pool, Device Farm assegnerà automaticamente quel test a quell'istanza del dispositivo. Se l'istanza del dispositivo non è disponibile, Device Farm metterà in coda il test sul dispositivo non appena sarà disponibile.

Creazione di un pool di dispositivi privato con dispositivi privati (console)

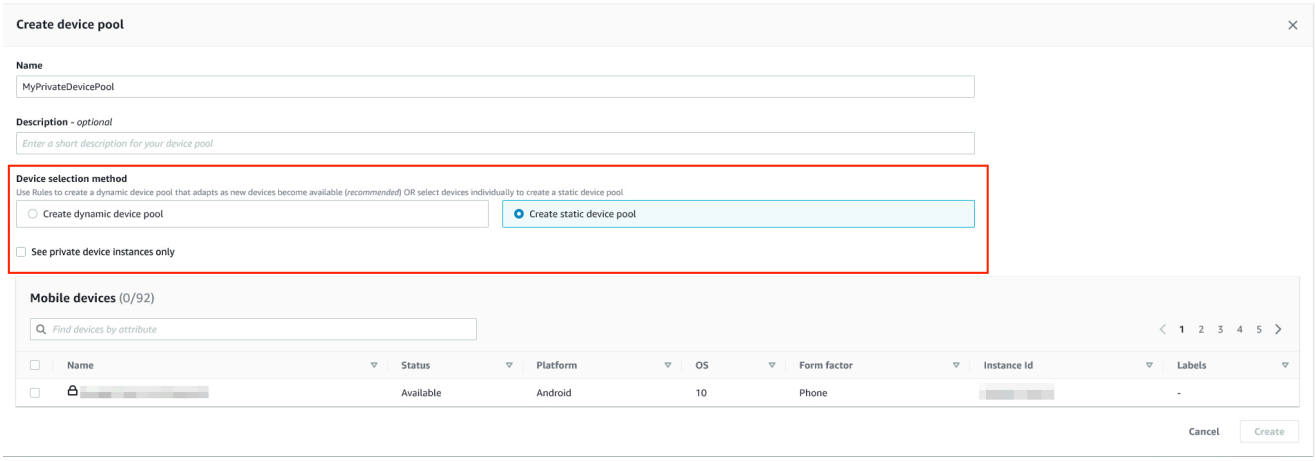
Quando si crea una sessione di test, è possibile creare un pool di dispositivi per la sessione di test e per assicurare che il pool includa solo i propri dispositivi privati.

> **Note**
>
> Quando si crea un pool di dispositivi con dispositivi privati nella console, è possibile utilizzare solo una delle tre regole disponibili per la selezione dei dispositivi privati. Se desideri creare un pool di dispositivi che contenga più tipi di regole per dispositivi privati (ad esempio, pool di dispositivi che contengono regole per il dispositivo ARNs e l'istanza del dispositivo ARNs), devi creare il pool tramite la CLI o l'API.

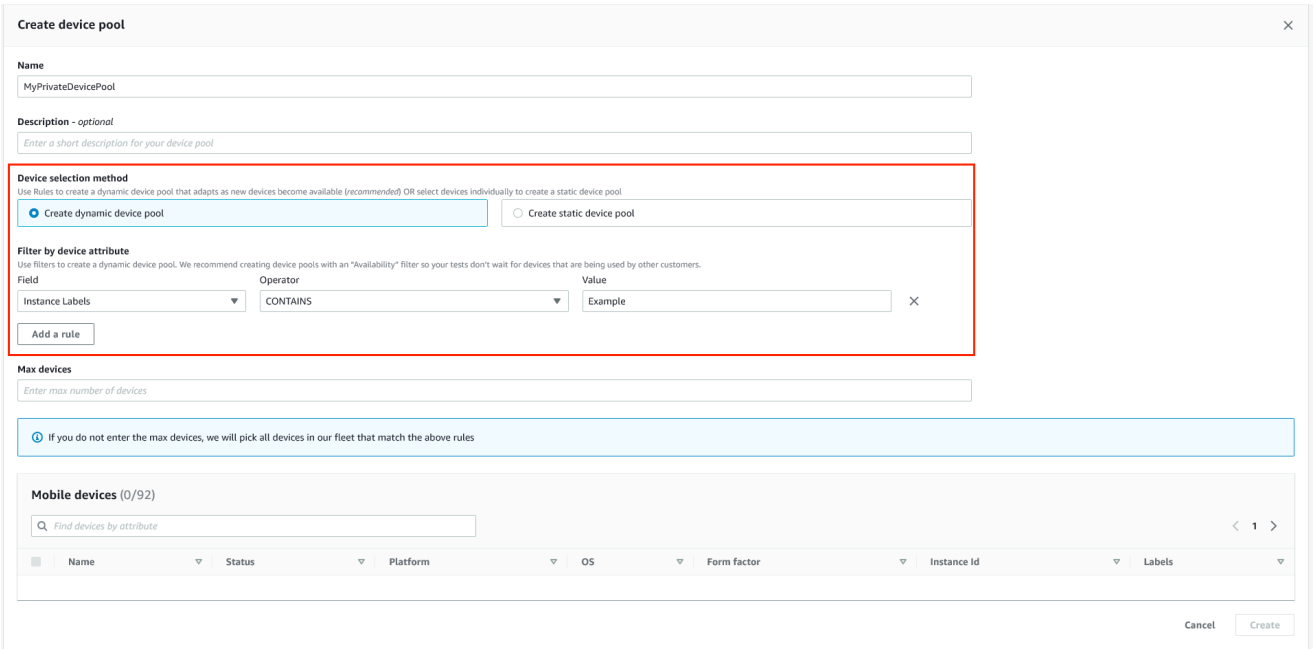
1. Apri la console Device Farm all'indirizzo https://console.aws.amazon.com/devicefarm/.
2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.

3. Scegli un progetto esistente dall'elenco o creane uno nuovo. Per creare un nuovo progetto, scegliete Nuovo progetto, inserite un nome per il progetto, quindi scegliete Invia.
4. Scegliete Impostazioni del progetto, quindi passate alla scheda Device pool.
5. Scegli Crea pool di dispositivi e inserisci un nome e una descrizione opzionale per il tuo pool di dispositivi.

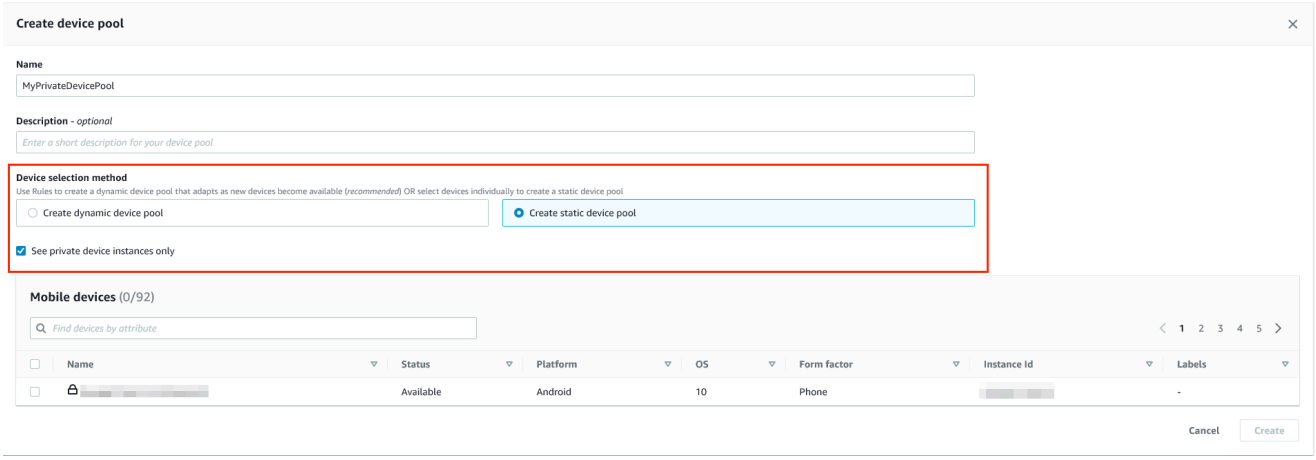
    a. Per utilizzare le regole ARN dei dispositivi per il tuo pool di dispositivi, scegli Crea pool di dispositivi statico, quindi seleziona i tipi di dispositivi specifici dall'elenco che desideri utilizzare nel pool di dispositivi. Non selezionate solo istanze di dispositivo private perché questa opzione fa sì che il pool di dispositivi venga creato con le regole ARN delle istanze del dispositivo (anziché le regole ARN del dispositivo).



    b. Per utilizzare le regole di etichettatura delle istanze di dispositivo per il tuo pool di dispositivi, scegli Crea pool dinamico di dispositivi. Quindi, per ogni etichetta che desideri utilizzare nel pool di dispositivi, scegli Aggiungi una regola. Per ogni regola, scegli Etichette di istanza come`Field`, scegli Contiene come e specifica l'`Operator`etichetta dell'istanza del dispositivo desiderata come`Value`.



c. Per utilizzare le regole ARN delle istanze di dispositivo per il tuo pool di dispositivi, scegli Crea pool di dispositivi statico, quindi seleziona Solo istanze di dispositivo private per limitare l'elenco dei dispositivi alle sole istanze di dispositivi privati che Device Farm ha associato al tuo account. AWS



6. Scegli Create (Crea).

Creazione di un pool di dispositivi privato con dispositivi privati ()AWS CLI

- Esegui il comando create-device-pool.

Per informazioni sull'utilizzo di Device Farm con AWS CLI, vedereAWS CLI riferimento.

Creazione di un pool di dispositivi privati con dispositivi privati (API)

- Chiamata dell'API `CreateDevicePool`.

Per informazioni sull'utilizzo dell'API Device Farm, vedereAutomazione di Device Farm.

Ignorare la nuova firma delle app su dispositivi privati in AWS Device Farm

La firma delle app è un processo che prevede la firma digitale di un pacchetto di app (ad esempio APK, IPA) con una chiave privata prima che possa essere installato su un dispositivo o pubblicato su un app store come Google Play Store o Apple App Store. Per semplificare i test riducendo il numero di firme e profili necessari e aumentare la sicurezza dei dati sui dispositivi remoti, AWS Device Farm firmerà nuovamente l'app dopo averla caricata sul servizio.

Una volta caricata l'app su AWS Device Farm, il servizio genererà una nuova firma per l'app utilizzando i propri certificati di firma e profili di provisioning. Questo processo sostituisce la firma originale dell'app con la firma di AWS Device Farm. L'app rifirmata viene quindi installata sui dispositivi di test forniti da AWS Device Farm. La nuova firma consente all'app di essere installata ed eseguita su questi dispositivi senza la necessità dei certificati originali dello sviluppatore.

Su iOS, sostituiamo il profilo di provisioning integrato con un profilo wildcard e firmiamo nuovamente l'app. Se lo fornisci, aggiungeremo dati ausiliari al pacchetto dell'applicazione prima dell'installazione, in modo che i dati siano presenti nella sandbox dell'app. La nuova firma dell'app iOS comporta la rimozione di tutti i diritti.

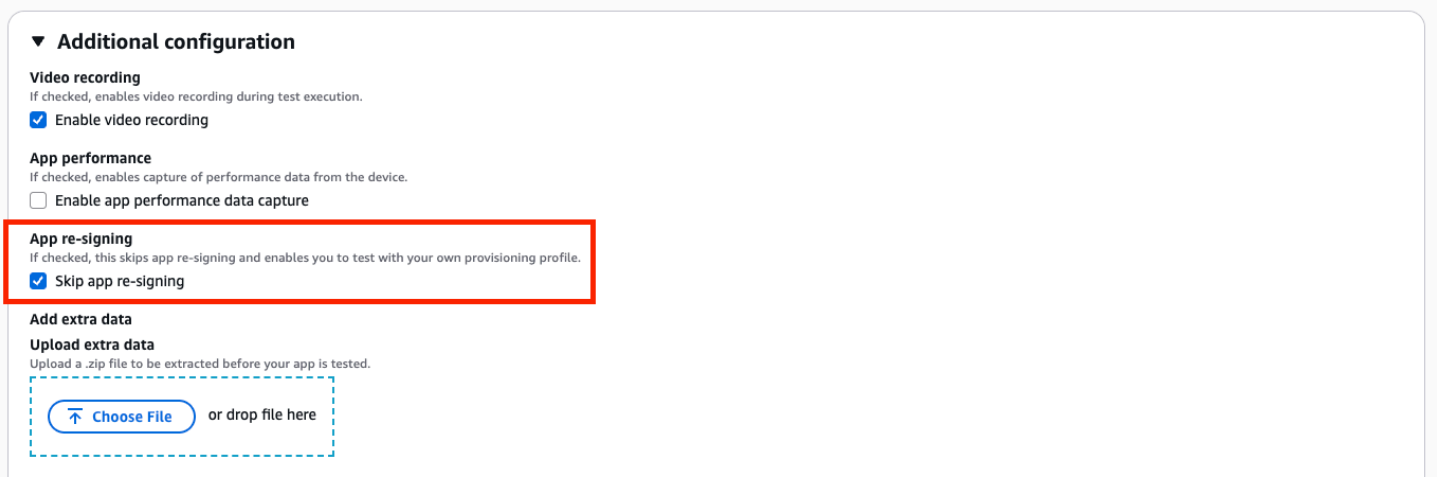
Su Android, firmiamo nuovamente l'app. Ciò potrebbe interrompere funzionalità che dipendono dalla firma dell'app, come l'API Android di Google Maps. Può inoltre attivare il rilevamento antipirateria e antimanomissione disponibile in prodotti come. DexGuard Per i test integrati, possiamo modificare il manifesto per includere le autorizzazioni necessarie per acquisire e salvare schermate.

Quando usi dispositivi privati, puoi saltare la fase in cui AWS Device Farm rifirma la tua app. Questo è diverso dai dispositivi pubblici, in cui Device Farm riassegna sempre la firma dell'app sulle piattaforme Android e iOS.

È possibile ignorare la nuova firma dell’app al momento della creazione di una sessione di accesso remoto o di test. Questo può essere utile se la funzionalità dell'app si interrompe quando Device

Farm firma nuovamente l'app. Ad esempio, le notifiche push potrebbero non funzionare dopo la nuova firma. Per ulteriori informazioni sulle modifiche apportate da Device Farm durante il test dell'app, consulta la pagina AWS Device Farm FAQs o Apps.

Per saltare la nuova firma dell'app per un'esecuzione di test, seleziona Ignora la nuova firma dell'app in Configurazione aggiuntiva. Questa opzione è disponibile solo per i dispositivi privati.



> **Note**
>
> Se utilizzi il XCTest framework, l'opzione Salta nuovamente la firma dell'app non è disponibile. Per ulteriori informazioni, consulta Integrazione di Device Farm con XCTest iOS.

Le fasi aggiuntive per configurare le impostazioni della firma delle app variano in base all'utilizzo di dispositivi privati Android o iOS.

Ignorare la nuova firma dell'app sui dispositivi Android

Se stai testando l'app su un dispositivo privato Android, seleziona Skip app re-signing (Ignora nuova firma dell'app) quando crei una sessione di test o di accesso remoto. Non è richiesta alcuna configurazione aggiuntiva.

Ignorare la nuova firma dell'app sui dispositivi iOS

Apple richiede di firmare un'app per l'esecuzione di test prima di caricarla su un dispositivo. Per i dispositivi iOS sono disponibili due opzioni per la firma dell'app.

- Se si sta utilizzando un profilo per sviluppatori interno (Enterprise), è possibile passare alla sezione successiva the section called “Crea una sessione di accesso remoto per rendere affidabile la tua app”.

- Se utilizzi un profilo di sviluppo di app iOS ad hoc, è necessario prima registrare il dispositivo con il tuo account sviluppatore Apple e aggiornare il profilo di provisioning in modo che includa il dispositivo privato. È necessario quindi firmare nuovamente la tua app con il profilo di provisioning che hai aggiornato. Puoi quindi eseguire l'app rifirmata in Device Farm.

Come eseguire la registrazione di un dispositivo con un profilo di provisioning di sviluppo di app iOS ad hoc

1. Eseguire l'accesso al proprio account sviluppatori Apple.
2. Vai alla sezione Certificati e profili della console. IDs
3. Vai a Devices (Dispositivi).
4. Registrare il dispositivo nel proprio account sviluppatore Apple. Per ottenere il nome e l'UDID del dispositivo, utilizza il `ListDeviceInstances` funzionamento dell'API Device Farm.
5. Andare al proprio profilo di provisioning e scegliere Edit (Modifica).
6. Scegliere il dispositivo dall'elenco.
7. In XCode, recuperare il proprio profilo di provisioning aggiornato e firmare nuovamente l'app.

Non è richiesta alcuna configurazione aggiuntiva. Ora puoi creare una sessione di accesso remoto o di test e selezionare Skip app re-signing (Ignora nuova firma dell'app).

Creazione di una sessione di accesso remoto per fidarsi della tua app iOS

Se utilizzi un profilo di provisioning per sviluppatori in-house (enterprise), devi eseguire una sola volta la procedura di conferma dell'attendibilità del certificato dello sviluppatore dell'app in-house su ciascuno dei tuoi dispositivi privati.

A tale scopo, devi installare un'app segnaposto firmata con lo stesso certificato dell'app che desideri testare. Dopo che il dispositivo ha considerato attendibile il profilo di configurazione o lo sviluppatore dell'app aziendale, tutte le app di quello sviluppatore sono considerate attendibili sul dispositivo privato fino a quando non vengono eliminate. Pertanto, quando installi nuove versioni dell'app che desideri testare, non dovrai fidarti nuovamente dello sviluppatore dell'app ogni volta. Ciò è

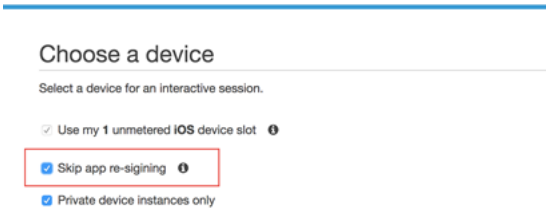
particolarmente utile se esegui automazioni dei test e non desideri creare una sessione di accesso remoto ogni volta che testi la tua app.

Una procedura comune utilizzata da molti clienti è quella di firmare nuovamente l'app di esempio Device Farm per iOS, quindi installarla sul proprio dispositivo come app segnaposto.

Prima di iniziare la sessione di accesso remoto, segui i passaggi indicati Creazione di un profilo di istanza in AWS Device Farm per creare o modificare un profilo di istanza in Device Farm. Nel profilo dell'istanza, aggiungi l'ID del pacchetto dell'app segnaposto all'impostazione Escludi pacchetti dalla pulizia. Quindi, collega il profilo dell'istanza all'istanza del dispositivo privato per assicurarti che Device Farm non rimuova l'app dal dispositivo prima che inizi una nuova esecuzione di test. In questo modo, il certificato dello sviluppatore rimane attendibile.

Puoi caricare l'app segnaposto sul dispositivo utilizzando una sessione di accesso remoto, che ti consente di avviare l'app e fidarti dello sviluppatore.

1. Seguire le istruzioni in Creazione di una sessione per creare una sessione di accesso remoto utilizzando il profilo dell'istanza del dispositivo privato creato. Quando si crea la sessione, assicurarsi di selezionare Skip app re-signing (Ignora rifirma dell'app).



> **Important**
>
> Per filtrare l'elenco di dispositivi in modo che includa solo i dispositivi privati, selezionare Private device instances only (Solo istanze di dispositivi privati) per assicurare che venga utilizzato un dispositivo privato con il corretto profilo dell'istanza.

   Assicurati di aggiungere anche l'app segnaposto o l'app che desideri testare all'impostazione Escludi pacchetti dalla pulizia per il profilo dell'istanza collegato a questa istanza.
2. All'avvio della sessione remota, scegli Scegli file per installare un'applicazione che utilizza il tuo profilo di provisioning interno.
3. Avvia l'app che hai appena caricato.

4. Verifica che venga visualizzata una finestra di dialogo iOS che indica che lo sviluppatore dell'app aziendale non è attendibile.
5. Quindi, se il dispositivo iOS utilizza la versione iOS 18 o successiva, apri un ticket di assistenza con il team di AWS Device Farm per fare in modo che il nostro team si fidi dell'app per te, poiché questi dispositivi richiedono che l'app sia affidabile manualmente. Altrimenti, se la versione iOS è 17 o precedente, puoi accedere all'app Impostazioni e, in Impostazioni generali, fidarti dell'app dal menu VPN e Profili.

Tutte le app di questo profilo di configurazione o di questo sviluppatore di app enterprise sono ora considerate attendibili su questo dispositivo privato finché non le elimini.

Amazon VPC in tutte le AWS regioni in AWS Device Farm

I servizi Device Farm sono disponibili solo nella regione Stati Uniti occidentali (Oregon) (`us-west-2`). Puoi usare Amazon Virtual Private Cloud (Amazon VPC) per raggiungere un servizio nel tuo Amazon Virtual Private Cloud in un'altra AWS regione utilizzando Device Farm. Se Device Farm e il tuo servizio si trovano nella stessa regione, vediUtilizzo dei servizi endpoint Amazon VPC con Device Farm - Legacy (non consigliato).

Esistono due modi per accedere ai servizi privati situati in una regione diversa. Se disponi di servizi situati in un'altra regione diversa`us-west-2`, puoi utilizzare VPC Peering per peerizzare il VPC di quella regione su un altro VPC che si interfaccia con Device Farm in. `us-west-2` Tuttavia, se disponi di servizi in più regioni, un Transit Gateway ti consentirà di accedere a tali servizi con una configurazione di rete più semplice.

Per ulteriori informazioni, consulta gli scenari di peering VPC nella Amazon VPC Peering Guide.

Panoramica del peering VPC per diverse regioni VPCs in AWS Device Farm

Puoi peerizzarne due VPCs in regioni diverse purché abbiano blocchi CIDR distinti e non sovrapposti. Ciò garantisce che tutti gli indirizzi IP privati siano unici e consente a tutte le risorse presenti di VPCs indirizzarsi tra loro senza la necessità di alcuna forma di traduzione degli indirizzi di rete (NAT). Per ulteriori informazioni sulla notazione CIDR, consulta RFC 4632.

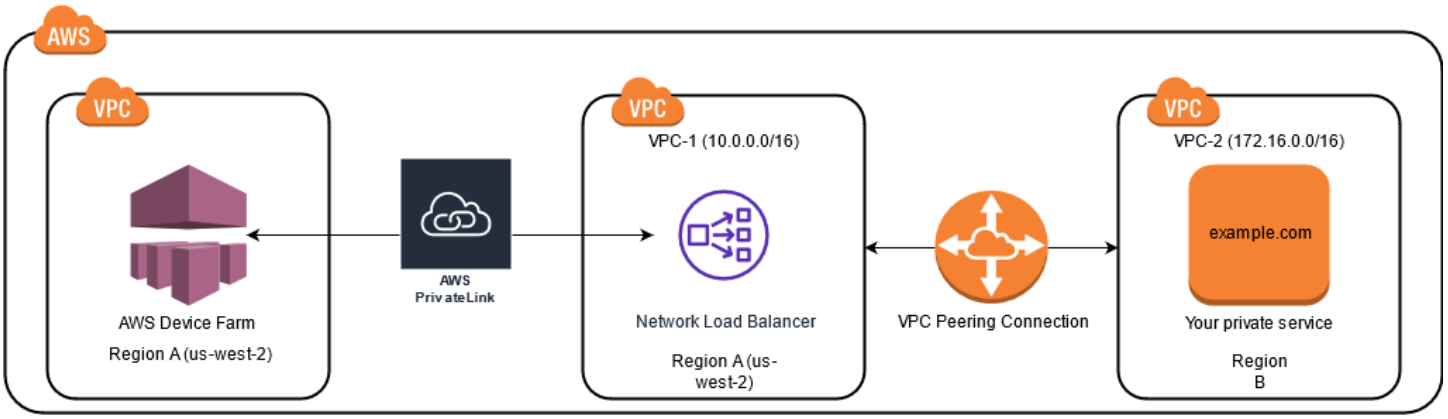
Questo argomento include uno scenario di esempio interregionale in cui Device Farm (denominata VPC-1) si trova nella regione Stati Uniti occidentali (Oregon) (). `us-west-2` Il secondo VPC in questo esempio (denominato VPC-2) si trova in un'altra regione.

Esempio di Device Farm VPC interregionale

Componente VPC	VPC-1	VPC-2
CIDR	10.0.0.0/16	172.16.0.0/16

> **Important**
>
> La creazione di una connessione peering tra due VPCs può modificare il livello di sicurezza di. VPCs Inoltre, l'aggiunta di nuove voci alle relative tabelle di routing può modificare il livello di sicurezza delle risorse all'interno di. VPCs È responsabilità dell'utente implementare queste configurazioni in modo da soddisfare i requisiti di sicurezza dell'organizzazione. Per ulteriori informazioni, consulta il modello di responsabilità condivisa.

Il seguente diagramma mostra i componenti nell'esempio e le interazioni tra i componenti.



Argomenti

- Prerequisiti per l'utilizzo di Amazon VPC in AWS Device Farm
- Fase 1: Configurazione di una connessione peering tra VPC-1 e VPC-2
- Fase 2: Aggiornamento delle tabelle di routing in VPC-1 e VPC-2
- Fase 3: Creazione di un gruppo target
- Fase 4: Creazione di un Network Load Balancer
- Fase 5: Creazione di un servizio endpoint VPC per connettere il VPC a Device Farm
- Fase 6: Creare una configurazione di endpoint VPC tra il VPC e Device Farm
- Passaggio 7: creazione di un'esecuzione di test per utilizzare la configurazione dell'endpoint VPC

- Creazione di una rete scalabile con Transit Gateway

Prerequisiti per l'utilizzo di Amazon VPC in AWS Device Farm

Questo esempio richiede quanto segue:

- Due VPCs configurate con sottoreti contenenti blocchi CIDR non sovrapposti.
- VPC-1 deve trovarsi nella `us-west-2` regione e contenere sottoreti per le zone `us-west-2a` di disponibilità e. `us-west-2b` `us-west-2c`

Per ulteriori informazioni sulla creazione VPCs e la configurazione di sottoreti, consulta Working with VPCs and subnet nella Amazon VPC Peering Guide.

Fase 1: Configurazione di una connessione peering tra VPC-1 e VPC-2

Stabilire una connessione peering tra i due blocchi CIDR VPCs contenenti blocchi CIDR non sovrapposti. A tale scopo, consulta Creare e accettare connessioni peering VPC nella Amazon VPC Peering Guide. Utilizzando lo scenario interregionale di questo argomento e la Amazon VPC Peering Guide, viene creato il seguente esempio di configurazione di connessione peering:

Nome

```
Device-Farm-Peering-Connection-1
```

ID VPC (richiedente)

```
vpc-0987654321gfedcba (VPC-2)
```

Account

```
My account
```

Region

```
US West (Oregon) (us-west-2)
```

ID VPC (Accetta)

```
vpc-1234567890abcdefg (VPC-1)
```

> **Note**
>
> Assicurati di consultare le quote di connessione peering VPC quando stabilisci nuove connessioni peering. Per ulteriori informazioni, consulta le quote di Amazon VPC nella Amazon VPC Peering Guide.

Fase 2: Aggiornamento delle tabelle di routing in VPC-1 e VPC-2

Dopo aver impostato una connessione peering, è necessario stabilire un percorso di destinazione tra i due VPCs per trasferire i dati tra di loro. Per stabilire questa route, è possibile aggiornare manualmente la tabella delle rotte di VPC-1 in modo che punti alla sottorete di VPC-2 e viceversa. A tale scopo, consulta Aggiornare le tabelle di routing per una connessione peering VPC nella Amazon VPC Peering Guide. Utilizzando lo scenario interregionale di questo argomento e la Amazon VPC Peering Guide, viene creato il seguente esempio di configurazione della tabella di routing:

Esempio di tabella di routing VPC di Device Farm

Componente VPC	VPC-1	VPC-2
ID della tabella di routing	rtb-1234567890abcdefg	rtb-0987654321gfedcba
Intervallo di indirizzi locali	10.0.0.0/16	172.16.0.0/16
Intervallo di indirizzi di destinazione	172.16.0.0/16	10.0.0.0/16

Fase 3: Creazione di un gruppo target

Dopo aver impostato i percorsi di destinazione, è possibile configurare un Network Load Balancer in VPC-1 per indirizzare le richieste a VPC-2.

Il Network Load Balancer deve innanzitutto contenere un gruppo target contenente gli indirizzi IP a cui vengono inviate le richieste.

Per creare un gruppo target

1. Identifica gli indirizzi IP del servizio che desideri utilizzare come target in VPC-2.
   - Questi indirizzi IP devono essere membri della sottorete utilizzata nella connessione peering.

- Gli indirizzi IP di destinazione devono essere statici e immutabili. Se il tuo servizio dispone di indirizzi IP dinamici, valuta la possibilità di indirizzare le richieste a una risorsa statica (ad esempio un Network Load Balancer) e di fare in modo che tale risorsa statica indirizzi le richieste verso la tua destinazione reale.

> **Note**
>
> - Se hai come target una o più istanze Amazon Elastic Compute Cloud (Amazon EC2) autonome, apri la console Amazon EC2 all'indirizzo, quindi scegli Istanze. https://console.aws.amazon.com/ec2/
> - Se hai come target un gruppo Amazon EC2 Auto Scaling di istanze Amazon EC2, devi associare il gruppo Amazon EC2 Auto Scaling a un Network Load Balancer. Per ulteriori informazioni, consulta Come collegare un bilanciatore del carico al gruppo Auto Scaling nella Guida per l'utente di Amazon EC2 Auto Scaling.
>
>   Quindi, puoi aprire la console Amazon EC2 all'indirizzo https://console.aws.amazon.com/ec2/, quindi scegliere Interfacce di rete. Da lì puoi visualizzare gli indirizzi IP per ciascuna delle interfacce di rete di Network Load Balancer in ogni zona di disponibilità.

2. Crea un gruppo target in VPC-1. A tale scopo, consulta Creare un gruppo target per il Network Load Balancer nella Guida per l'utente di Network Load Balancer.

   I gruppi target per i servizi in un VPC diverso richiedono la seguente configurazione:

   - Per Scegli un tipo di destinazione, scegli gli indirizzi IP.
   - Per VPC, scegli il VPC che ospiterà il load balancer. Per l'esempio dell'argomento, questo sarà VPC-1.
   - Nella pagina Registra destinazioni, registra una destinazione per ogni indirizzo IP in VPC-2.

     Per Rete, scegli Altro indirizzo IP privato.

     Per Zona di disponibilità, scegli le zone desiderate in VPC-1.

     Per IPv4 l'indirizzo, scegli l'indirizzo IP VPC-2.

     Per Porte, scegli le tue porte.
   - Seleziona Includi come in sospeso di seguito. Quando hai finito di specificare gli indirizzi, scegli Registra obiettivi in sospeso.

Utilizzando lo scenario interregionale di questo argomento e la Guida utente per Network Load Balancers, nella configurazione del gruppo target vengono utilizzati i seguenti valori:

Target type (Tipo di destinazione)

```
IP addresses
```

Nome del gruppo target

```
my-target-group
```

Protocollo/porta

```
TCP : 80
```

VPC

```
vpc-1234567890abcdefg (VPC-1)
```

Rete

```
Other private IP address
```

Zona di disponibilità

```
all
```

IPv4 address

```
172.16.100.60
```

Porte

```
80
```

Fase 4: Creazione di un Network Load Balancer

Creare un Network Load Balancer utilizzando il gruppo target descritto nel passaggio 3. A tale scopo, vedere Creazione di un Network Load Balancer.

Utilizzando lo scenario interregionale di questo argomento, i seguenti valori vengono utilizzati in una configurazione di esempio di Network Load Balancer:

Load balancer name (Nome del sistema di bilanciamento del carico)

```
my-nlb
```

Schema

```
Internal
```

VPC

```
vpc-1234567890abcdefg (VPC-1)
```

Mappatura

```
us-west-2a - subnet-4i23iuufkdiufsloi

us-west-2b - subnet-7x989pkjj78nmn23j

us-west-2c - subnet-0231ndmas12bnnsds
```

Protocollo/porta

```
TCP : 80
```

Gruppo bersaglio

```
my-target-group
```

Fase 5: Creazione di un servizio endpoint VPC per connettere il VPC a Device Farm

Puoi utilizzare Network Load Balancer per creare un servizio endpoint VPC. Tramite questo servizio endpoint VPC, Device Farm può connettersi al servizio in VPC-2 senza alcuna infrastruttura aggiuntiva, come un gateway Internet, un'istanza NAT o una connessione VPN.

A tale scopo, consulta Creazione di un servizio endpoint Amazon VPC.

Fase 6: Creare una configurazione di endpoint VPC tra il VPC e Device Farm

Ora puoi stabilire una connessione privata tra il tuo VPC e Device Farm. È possibile utilizzare Device Farm per testare servizi privati senza esporli attraverso la rete Internet pubblica. A tale scopo, consulta Creazione di una configurazione di endpoint VPC in Device Farm.

Utilizzando lo scenario interregionale di questo argomento, i seguenti valori vengono utilizzati in una configurazione di endpoint VPC di esempio:

Nome

```
My VPCE Configuration
```

Nome del servizio VPCE

```
com.amazonaws.vpce.us-west-2.vpce-svc-1234567890abcdefg
```

Nome DNS del servizio

```
devicefarm.com
```

Passaggio 7: creazione di un'esecuzione di test per utilizzare la configurazione dell'endpoint VPC

È possibile creare esecuzioni di test che utilizzano la configurazione degli endpoint VPC descritta nel passaggio 6. Per ulteriori informazioni, consulta Creazione di un'esecuzione di test in Device Farm o Creazione di una sessione.

Creazione di una rete scalabile con Transit Gateway

Per creare una rete scalabile utilizzandone più di due VPCs, puoi utilizzare Transit Gateway per fungere da hub di transito di rete per interconnettere le tue reti VPCs e quelle locali. Per configurare un VPC nella stessa regione di Device Farm per utilizzare un Transit Gateway, puoi seguire la guida Amazon VPC endpoint services with Device Farm per indirizzare le risorse in un'altra regione in base ai loro indirizzi IP privati.

Per ulteriori informazioni su Transit Gateway, consulta Cos'è un gateway di transito? nella Amazon VPC Transit Gateways Guide.

Disattivazione di dispositivi privati in Device Farm

<Per disattivare un dispositivo privato dopo il periodo iniziale concordato, è n
Per ulteriori informazioni sui dispositivi privati, consulta. Dispositivi privati in AWS Device Farm

> **Important**
>
> Queste istruzioni si applicano solo alla risoluzione dei contratti relativi ai dispositivi privati. Per tutti gli altri AWS servizi e problemi di fatturazione, consulta la documentazione relativa a tali prodotti o contatta AWS l'assistenza.

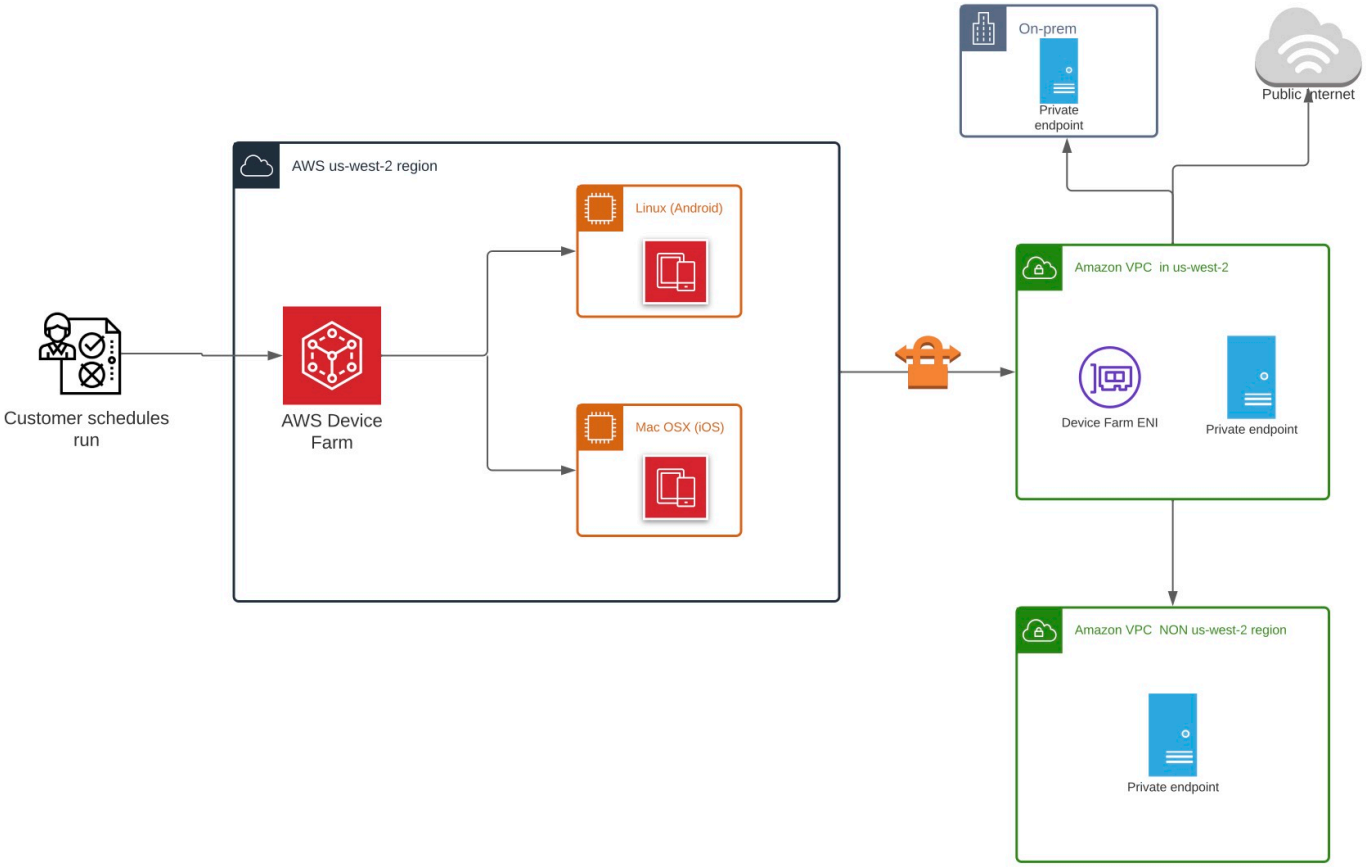
VPC-ENI nella Device Farm di AWS

> **Warning**
>
> Questa funzionalità è disponibile solo su dispositivi privati. Per richiedere l'uso privato del dispositivo sul tuo AWS account, contattaci. Se hai già aggiunto dispositivi privati al tuo AWS account, ti consigliamo vivamente di utilizzare questo metodo di connettività VPC.

La funzionalità di connettività VPC-ENI di AWS Device Farm aiuta i clienti a connettersi in modo sicuro ai propri endpoint privati ospitati su software on-premise o su AWS un altro provider cloud.

Puoi connettere sia i dispositivi mobili Device Farm che le relative macchine host a un ambiente Amazon Virtual Private Cloud (Amazon VPC) nella `us-west-2` regione, che consente l'accesso a non-internet-facing servizi e applicazioni isolati tramite un'interfaccia di rete elastica. Per ulteriori informazioni VPCs, consulta la Amazon VPC User Guide.

Se il tuo endpoint o VPC privato non si trova `us-west-2` nella regione, puoi collegarlo a un VPC `us-west-2` della regione utilizzando soluzioni come Transit Gateway o VPC Peering. In tali situazioni, Device Farm creerà un ENI in una sottorete fornita per il VPC `us-west-2` della regione e sarete responsabili di garantire che sia possibile stabilire una connessione tra il VPC `us-west-2` della regione e il VPC dell'altra regione.



Per informazioni sull'utilizzo della funzione AWS CloudFormation di creazione automatica e peer VPCs, consulta i VPCPeering modelli nell'archivio dei modelli su. AWS CloudFormation GitHub

> **Note**
>
> Device Farm non addebita alcun costo per la creazione ENIs nel VPC di un cliente. `us-west-2` Il costo per la connettività inter-VPC esterna o interregionale non è incluso in questa funzionalità.

Una volta configurato l'accesso al VPC, i dispositivi e le macchine host che utilizzi per i test non saranno in grado di connettersi a risorse esterne al VPC (ad esempio pubbliche CDNs) a meno che

non sia presente un gateway NAT specificato all'interno del VPC. Per ulteriori informazioni, consulta Gateway NAT nella Guida per l'utente di Amazon VPC.

Argomenti

AWS controllo degli accessi e IAM

AWS Device Farm consente di utilizzare AWS Identity and Access Management(IAM) per creare policy che garantiscono o limitano l'accesso alle funzionalità di Device Farm. Per utilizzare la funzionalità di connettività VPC con AWS Device Farm, è richiesta la seguente policy IAM per l'account utente o il ruolo che utilizzi per accedere ad AWS Device Farm:

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"
        }
      }
    }
  ]
}
```

Per creare o aggiornare un progetto Device Farm con una configurazione VPC, la policy IAM deve consentire di eseguire le seguenti azioni sulle risorse elencate nella configurazione VPC:

```
"ec2:DescribeVpcs"
"ec2:DescribeSubnets"
"ec2:DescribeSecurityGroups"
"ec2:CreateNetworkInterface"
```

Inoltre, la tua policy IAM deve consentire anche la creazione del ruolo collegato al servizio:

```
"iam:CreateServiceLinkedRole"
```

> **Note**
>
> Nessuna di queste autorizzazioni è richiesta per gli utenti che non utilizzano configurazioni VPC nei propri progetti.

Ruoli collegati ai servizi

AWS Device Farm utilizza AWS Identity and Access Management ruoli collegati ai servizi (IAM). Un ruolo collegato ai servizi è un tipo unico di ruolo IAM collegato direttamente a Device Farm. I ruoli collegati ai servizi sono predefiniti da Device Farm e includono tutte le autorizzazioni richieste dal servizio per chiamare altri AWS servizi per conto dell'utente.

Un ruolo collegato al servizio semplifica la configurazione di Device Farm perché non è necessario aggiungere manualmente le autorizzazioni necessarie. Device Farm definisce le autorizzazioni dei

suoi ruoli collegati ai servizi e, se non diversamente definito, solo Device Farm può assumerne i ruoli. Le autorizzazioni definite includono la policy di attendibilità e la policy delle autorizzazioni che non può essere allegata a nessun'altra entità IAM.

È possibile eliminare un ruolo collegato al servizio solo dopo avere eliminato le risorse correlate. In questo modo proteggi le tue risorse Device Farm perché non puoi rimuovere inavvertitamente l'autorizzazione ad accedere alle risorse.

Per informazioni sugli altri servizi che supportano i ruoli collegati ai servizi, consulta Servizi AWS supportati da IAM e cerca i servizi che riportano Sì nella colonna Ruolo associato ai servizi. Scegli un Sì con un link per visualizzare la documentazione relativa al ruolo collegato ai servizi per tale servizio.

Autorizzazioni di ruolo collegate ai servizi per Device Farm

Device Farm utilizza il ruolo collegato al servizio denominato AWSServiceRoleForDeviceFarm: consente a Device Farm di accedere alle risorse AWS per tuo conto.

Il ruolo AWSService RoleForDeviceFarm collegato al servizio prevede che i seguenti servizi assumano il ruolo:

- `devicefarm.amazonaws.com`

La politica di autorizzazione dei ruoli consente a Device Farm di completare le seguenti azioni:

- Per il tuo account
  - Crea interfacce di rete
  - Descrizione delle interfacce di rete
  - Descriva VPCs
  - Descrivi le sottoreti
  - Descrivere i gruppi di sicurezza
  - Eliminare le interfacce
  - Modificare le interfacce di rete
- Per interfacce di rete
  - Crea tag
- Per interfacce di rete EC2 gestite da Device Farm
  - Crea le autorizzazioni per le interfacce di rete

La policy IAM completa recita:

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"
   }
  }
```

```
  }
 ]
}
```

Per consentire a un'entità IAM (come un utente, un gruppo o un ruolo) di creare, modificare o eliminare un ruolo collegato al servizio è necessario configurare le relative autorizzazioni. Per ulteriori informazioni, consulta Autorizzazioni del ruolo collegato ai servizi nella Guida per l'utente di IAM.

Creazione di un ruolo collegato al servizio per Device Farm

Quando fornisci una configurazione VPC per un progetto di test mobile, non è necessario creare manualmente un ruolo collegato al servizio. Quando crei la tua prima risorsa Device Farm nella Console di gestione AWS AWS CLI, o nell' AWS API, Device Farm crea automaticamente il ruolo collegato al servizio.

Se elimini questo ruolo collegato al servizio, è possibile ricrearlo seguendo lo stesso processo utilizzato per ricreare il ruolo nell'account. Quando crei la tua prima risorsa Device Farm, Device Farm crea nuovamente il ruolo collegato al servizio per te.

Puoi anche utilizzare la console IAM per creare un ruolo collegato al servizio con lo use case Device Farm. Nell'API AWS CLI o nell' AWS API, crea un ruolo collegato al servizio con il nome del servizio. `devicefarm.amazonaws.com` Per ulteriori informazioni, consulta Creazione di un ruolo collegato ai servizi nella Guida per l'utente IAM. Se elimini il ruolo collegato ai servizi, è possibile utilizzare lo stesso processo per crearlo nuovamente.

Modifica di un ruolo collegato al servizio per Device Farm

Device Farm non consente di modificare il ruolo AWSService RoleForDeviceFarm collegato al servizio. Dopo avere creato un ruolo collegato al servizio, non sarà possibile modificarne il nome perché varie entità potrebbero farvi riferimento. È possibile tuttavia modificarne la descrizione utilizzando IAM. Per ulteriori informazioni, consulta la sezione Modifica di un ruolo collegato ai servizi nella Guida per l'utente di IAM.

Eliminazione di un ruolo collegato al servizio per Device Farm

Se non è più necessario utilizzare una funzionalità o un servizio che richiede un ruolo collegato al servizio, ti consigliamo di eliminare il ruolo. In questo modo non sarà più presente un'entità non

utilizzata che non viene monitorata e gestita attivamente. Tuttavia, è necessario effettuare la pulizia delle risorse associate al ruolo collegato al servizio prima di poterlo eliminare manualmente.

> **Note**
>
> Se il servizio Device Farm utilizza il ruolo quando si tenta di eliminare le risorse, l'eliminazione potrebbe non riuscire. In questo caso, attendi alcuni minuti e quindi ripeti l'operazione.

Per eliminare manualmente il ruolo collegato ai servizi mediante IAM

Utilizza la console IAM AWS CLI, il o l' AWS API per eliminare il ruolo AWSService RoleForDeviceFarm collegato al servizio. Per ulteriori informazioni, consulta Eliminazione del ruolo collegato al servizio nella Guida per l'utente di IAM.

Regioni supportate per i ruoli collegati ai servizi di Device Farm

Device Farm supporta l'utilizzo di ruoli collegati al servizio in tutte le regioni in cui il servizio è disponibile. Per ulteriori informazioni, consulta AWS Regioni ed endpoint.

Device Farm non supporta l'utilizzo di ruoli collegati al servizio in tutte le aree geografiche in cui il servizio è disponibile. È possibile utilizzare il AWSService RoleForDeviceFarm ruolo nelle seguenti regioni.

Nome della Regione	Identità della Regione	Support in Device Farm
Stati Uniti orientali (Virginia settentrionale)	us-east-1	No
Stati Uniti orientali (Ohio)	us-east-2	No
Stati Uniti occidentali (California settentrionale)	us-west-1	No
Stati Uniti occidentali (Oregon)	us-west-2	Sì
Asia Pacifico (Mumbai)	ap-south-1	No
Asia Pacifico (Osaka-Locale)	ap-northeast-3	No

Nome della Regione	Identità della Regione	Support in Device Farm
Asia Pacific (Seoul)	ap-northeast-2	No
Asia Pacifico (Singapore)	ap-southeast-1	No
Asia Pacifico (Sydney)	ap-southeast-2	No
Asia Pacifico (Tokyo)	ap-northeast-1	No
Canada (Centrale)	ca-central-1	No
Europa (Francoforte)	eu-central-1	No
Europa (Irlanda)	eu-west-1	No
Europe (London)	eu-west-2	No
Europa (Parigi)	eu-west-3	No
Sud America (San Paolo)	sa-east-1	No
AWS GovCloud (US)	us-gov-west-1	No

Prerequisiti

L'elenco seguente descrive alcuni requisiti e suggerimenti da esaminare durante la creazione di configurazioni VPC-ENI:

- I dispositivi privati devono essere assegnati al tuo account. AWS
- È necessario disporre di un AWS account, utente o ruolo con autorizzazioni per creare un ruolo collegato al servizio. Quando si utilizzano endpoint Amazon VPC con funzionalità di test mobile di Device Farm, Device Farm crea un ruolo collegato al servizio AWS Identity and Access Management (IAM).
- Device Farm può connettersi VPCs solo nella `us-west-2` regione. Se non disponi di un VPC nella `us-west-2` regione, devi crearne uno. Quindi, per accedere alle risorse in un VPC in un'altra regione, è necessario stabilire una connessione peering tra il VPC nella regione `us-west-2` e il VPC nell'altra regione. Per informazioni sul peering VPCs, consulta la Amazon VPC Peering Guide.

È necessario verificare di avere accesso al VPC specificato quando si configura la connessione. È necessario configurare determinate autorizzazioni Amazon Elastic Compute Cloud (Amazon EC2) per Device Farm.

- La risoluzione DNS è richiesta nel VPC che utilizzi.
- Una volta creato il VPC, avrai bisogno delle seguenti informazioni sul VPC della regione: `us-west-2`
  - ID VPC
  - Subnet IDs (solo sottoreti private)
  - Gruppo di sicurezza IDs
- È necessario configurare le connessioni Amazon VPC in base al progetto. Al momento, puoi configurare solo una configurazione VPC per progetto. Quando configuri un VPC, Amazon VPC crea un'interfaccia all'interno del tuo VPC e la assegna alle sottoreti e ai gruppi di sicurezza specificati. Tutte le sessioni future associate al progetto utilizzeranno la connessione VPC configurata.
- Non è possibile utilizzare le configurazioni VPC-ENI insieme alla funzionalità VPCE precedente.
- Consigliamo vivamente di non aggiornare un progetto esistente con una configurazione VPC-ENI poiché i progetti esistenti potrebbero avere impostazioni VPCE che persistono a livello di esecuzione. Invece, se utilizzi già le funzionalità VPCE esistenti, usa VPC-ENI per tutti i nuovi progetti.

Connessione ad Amazon VPC

Puoi configurare e aggiornare il tuo progetto per utilizzare gli endpoint Amazon VPC. La configurazione VPC-ENI è configurata per progetto. Un progetto può avere un solo endpoint VPC-ENI alla volta. Per configurare l'accesso VPC per un progetto, è necessario conoscere i seguenti dettagli:

- L'ID VPC è inserito `us-west-2` se l'app è ospitata lì o l'ID `us-west-2` VPC che si connette a un altro VPC in un'altra regione.
- I gruppi di sicurezza applicabili da applicare alla connessione.
- Le sottoreti che verranno associate alla connessione. All'avvio di una sessione, viene utilizzata la sottorete più grande disponibile. Ti consigliamo di associare più sottoreti a diverse zone di disponibilità per migliorare la posizione di disponibilità della connettività VPC.
- Quando si utilizza VPC-ENI, il resolver DNS utilizzato dagli host e dai dispositivi di test di Device Farm sarà il server fornito dai servizi DHCP nella sottorete del cliente. In una configurazione

predefinita, questo sarà il resolver predefinito del VPC. I clienti che desiderano specificare resolver DNS personalizzati possono configurare un set di opzioni DHCP nel proprio VPC.

Dopo aver creato la configurazione VPC-ENI, puoi aggiornarne i dettagli utilizzando la console o la CLI seguendo i passaggi seguenti.

Console

1. Accedere alla console Device Farm all'indirizzo https://console.aws.amazon.com/devicefarm.
2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.
3. In Mobile Testing projects, scegli il nome del tuo progetto dall'elenco.
4. Selezionare Project settings (Impostazioni del progetto).
5. Nella sezione Impostazioni Virtual Private Cloud (VPC), puoi modificareVPC, `Subnets` (solo sottoreti private) e. `Security Groups`
6. Scegli Save (Salva).

CLI

Utilizza il seguente comando AWS CLI per aggiornare Amazon VPC:

```
$  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
```

Puoi anche configurare un Amazon VPC durante la creazione del tuo progetto:

```
$  aws devicefarm create-project \
--name VPCDemo \
--vpc-config \
securityGroupIds=sg-02c1537701a7e3763,sg-005dadf9311efda25,\
subnetIds=subnet-09b1a45f9cac53717,subnet-09b1a45f9cac12345,\
vpcId=vpc-0238fb322af81a368
```

Limits

Le seguenti limitazioni sono applicabili alla funzionalità VPC-ENI:

- Puoi fornire fino a cinque gruppi di sicurezza nella configurazione VPC di un progetto Device Farm.
- È possibile fornire fino a otto sottoreti nella configurazione VPC di un progetto Device Farm.
- Quando configuri un progetto Device Farm per utilizzarlo con il tuo VPC, la sottorete più piccola che puoi fornire deve avere almeno cinque indirizzi disponibili. IPv4
- Al momento gli indirizzi IP pubblici non sono supportati. Ti consigliamo invece di utilizzare sottoreti private nei tuoi progetti Device Farm. Se hai bisogno di un accesso pubblico a Internet durante i test, utilizza un gateway NAT (Network Address Translation). La configurazione di un progetto Device Farm con una sottorete pubblica non fornisce ai test l'accesso a Internet o un indirizzo IP pubblico.
- L'integrazione VPC-ENI supporta solo sottoreti private nel tuo VPC.
- È supportato solo il traffico in uscita dall'ENI gestito dal servizio. Ciò significa che l'ENI non può ricevere richieste in entrata non richieste dal VPC.

Utilizzo dei servizi endpoint Amazon VPC con Device Farm - Legacy (non consigliato)

Warning

Consigliamo vivamente di utilizzare la connettività VPC-ENI descritta in questa pagina per la connettività endpoint privata poiché VPCE è ora considerata una funzionalità legacy. VPC-ENI offre maggiore flessibilità, configurazioni più semplici, è più efficiente in termini di costi e richiede un sovraccarico di manutenzione significativamente inferiore rispetto al metodo di connettività VPCE.

Note

L'utilizzo di Amazon VPC Endpoint Services con Device Farm è supportato solo per i clienti con dispositivi privati configurati. Per consentire al tuo account AWS di utilizzare questa funzionalità con dispositivi privati, contattaci.

Amazon Virtual Private Cloud (Amazon VPC) è un AWS servizio che puoi utilizzare per avviare AWS risorse in una rete virtuale definita dall'utente. Con un VPC, hai il controllo sulle impostazioni di rete, come l'intervallo di indirizzi IP, le sottoreti, le tabelle di routing e i gateway di rete.

Se utilizzi Amazon VPC per ospitare applicazioni private nella AWS regione Stati Uniti occidentali (Oregon) (`us-west-2`), puoi stabilire una connessione privata tra il tuo VPC e Device Farm. Con questa connessione, è possibile utilizzare Device Farm per testare applicazioni private senza esporle attraverso la rete Internet pubblica. Per consentire al tuo AWS account di utilizzare questa funzionalità con dispositivi privati, contattaci.

Per connettere una risorsa nel tuo VPC a Device Farm, puoi utilizzare la console Amazon VPC per creare un servizio endpoint VPC. Questo servizio endpoint ti consente di fornire la risorsa del tuo VPC a Device Farm tramite un endpoint VPC Device Farm. Il servizio endpoint fornisce una connettività affidabile e scalabile a Device Farm senza richiedere un gateway Internet, un'istanza NAT (Network Address Translation) o una connessione VPN. Per ulteriori informazioni, consulta VPC endpoint services (AWS PrivateLink) nella Guida.AWS PrivateLink

> **Important**
>
> La funzionalità endpoint VPC di Device Farm ti aiuta a connettere in modo sicuro i servizi interni privati del tuo VPC al VPC pubblico di Device Farm utilizzando le connessioni. AWS PrivateLink Anche se la connessione è sicura e privata, tale sicurezza dipende dalla protezione delle proprie credenziali AWS . Se AWS le tue credenziali sono compromesse, un utente malintenzionato può accedere ai dati del servizio o esporli al mondo esterno.

Dopo aver creato un servizio di endpoint VPC in Amazon VPC, puoi utilizzare la console Device Farm per creare una configurazione di endpoint VPC in Device Farm. Questo argomento mostra come creare la connessione Amazon VPC e la configurazione degli endpoint VPC in Device Farm.

Prima di iniziare

Le seguenti informazioni sono per gli utenti di Amazon VPC nella regione Stati Uniti occidentali (Oregon) (`us-west-2`), con una sottorete in ciascuna delle seguenti zone di disponibilità: us-west-2a, us-west-2b e us-west-2c.

Device Farm ha requisiti aggiuntivi per i servizi endpoint VPC con cui è possibile utilizzarlo. Quando crei e configuri un servizio endpoint VPC per funzionare con Device Farm, assicurati di scegliere opzioni che soddisfino i seguenti requisiti:

- Le zone di disponibilità per il servizio devono includere us-west-2a, us-west-2b e us-west-2c. Il Network Load Balancer associato a un servizio endpoint VPC determina le zone di disponibilità per quel servizio endpoint VPC. Se il tuo servizio endpoint VPC non mostra tutte e tre queste zone di disponibilità, devi ricreare il Network Load Balancer per abilitare queste tre zone e quindi riassociare Network Load Balancer al servizio endpoint.
- I principali consentiti per il servizio endpoint devono includere l'Amazon Resource Name (ARN) dell'endpoint VPC Device Farm (service ARN). Dopo aver creato il servizio endpoint, aggiungi l'ARN del servizio endpoint VPC Device Farm all'elenco degli utenti consentiti per concedere a Device Farm l'autorizzazione ad accedere al servizio endpoint VPC. Per ottenere l'ARN del servizio endpoint VPC Device Farm, contattaci.

Inoltre, se mantieni attiva l'impostazione Accettazione richiesta quando crei il servizio endpoint VPC, devi accettare manualmente ogni richiesta di connessione che Device Farm invia al servizio endpoint. Per modificare questa impostazione per un servizio endpoint esistente, scegli il servizio endpoint sulla console Amazon VPC, scegli Azioni, quindi scegli Modifica impostazione di accettazione degli endpoint. Per ulteriori informazioni, consulta Modifica dei sistemi di bilanciamento del carico e delle impostazioni di accettazione nella Guida.AWS PrivateLink

La sezione successiva spiega come creare un servizio endpoint Amazon VPC che soddisfi questi requisiti.

Fase 1: Creazione di un Network Load Balancer

Il primo passo per stabilire una connessione privata tra il VPC e Device Farm consiste nel creare un Network Load Balancer per indirizzare le richieste a un gruppo target.

New console

Per creare un Network Load Balancer utilizzando la nuova console

1. Apri la console Amazon Elastic Compute Cloud (Amazon EC2) all'indirizzo. https://console.aws.amazon.com/ec2/
2. Nel pannello di navigazione, in Load balancing, scegli Load balancers.
3. Selezionare Create Load Balancer (Crea sistema di bilanciamento del carico).
4. In Network load balancer, scegli Crea.
5. Nella pagina Crea sistema di bilanciamento del carico di rete, in Configurazione di base, procedi come segue:

a. Immettete il nome di un load balancer.

b. Per Schema, scegliete Interno.

6. In Network mappings (Mappature di rete), esegui le operazioni seguenti:

a. Scegli il VPC per il tuo gruppo target.

b. Seleziona le seguenti mappature:

- `us-west-2a`
- `us-west-2b`
- `us-west-2c`

7. In Listener and routing, utilizzate le opzioni Protocollo e Porta per scegliere il gruppo target.

> **Note**
>
> Per impostazione predefinita, il bilanciamento del carico tra zone di disponibilità è disabilitato.
> Poiché il sistema di bilanciamento del carico utilizza le zone `us-west-2a` di disponibilità e `us-west-2c` richiede la registrazione degli obiettivi in ciascuna di tali zone di disponibilità oppure, se si registrano i target in meno di tutte e tre le zone, è necessario abilitare il bilanciamento del carico tra zone. `us-west-2b` In caso contrario, il sistema di bilanciamento del carico potrebbe non funzionare come previsto.

8. Selezionare Create Load Balancer (Crea sistema di bilanciamento del carico).

Old console

Per creare un Network Load Balancer utilizzando la vecchia console

1. Apri la console Amazon Elastic Compute Cloud (Amazon EC2) all'indirizzo. https://console.aws.amazon.com/ec2/
2. Nel pannello di navigazione, in Load balancing, scegli Load balancer.
3. Selezionare Create Load Balancer (Crea sistema di bilanciamento del carico).
4. In Network load balancer, scegli Crea.

5. Nella pagina Configura il bilanciamento del carico, in Configurazione di base, procedi come segue:

   a. Immettete il nome di un load balancer.

   b. Per Schema, scegliete Interno.

6. In Listener, seleziona il protocollo e la porta utilizzati dal gruppo target.

7. In Zone di disponibilità, procedi come segue:

   a. Scegli il VPC per il tuo gruppo target.

   b. Seleziona le seguenti zone di disponibilità:

      - `us-west-2a`
      - `us-west-2b`
      - `us-west-2c`

   c. Scegli Avanti: configura le impostazioni di sicurezza.

8. (Facoltativo) Configura le impostazioni di sicurezza, quindi scegli Avanti: configura il routing.

9. Nella pagina Configure Routing (Configurazione dell'instradamento), procedere come segue:

   a. Per Target group (Gruppo di destinazione), scegliere Existing target group (Gruppo di destinazione esistente).

   b. Per Nome, scegli il tuo gruppo target.

   c. Scegli Avanti: registra gli obiettivi.

10. Nella pagina Registra obiettivi, esamina gli obiettivi, quindi scegli Avanti: revisione.

    > **Note**
    >
    > Per impostazione predefinita, il bilanciamento del carico tra zone di disponibilità è disabilitato.
    > Poiché il sistema di bilanciamento del carico utilizza le zone `us-west-2a` di disponibilità e `us-west-2c` richiede la registrazione degli obiettivi in ciascuna di tali zone di disponibilità oppure, se si registrano i target in meno di tutte e tre le zone, è necessario abilitare il bilanciamento del carico tra zone. `us-west-2b` In caso contrario, il sistema di bilanciamento del carico potrebbe non funzionare come previsto.

11. Controlla la configurazione del sistema di bilanciamento del carico, quindi scegli Crea.

Fase 2: creazione di un servizio endpoint Amazon VPC

Dopo aver creato il Network Load Balancer, usa la console Amazon VPC per creare un servizio endpoint nel tuo VPC.

1. Apri la console Amazon VPC all'indirizzo https://console.aws.amazon.com/vpc/.
2. In Risorse per regione, scegli Endpoint services.
3. Scegli Create Endpoint Service (Crea servizio endpoint).
4. Esegui una delle seguenti operazioni:
   - Se disponi già di un Network Load Balancer che desideri venga utilizzato dal servizio endpoint, selezionalo in Available load balancer, quindi continua con il passaggio 5.
   - Se non hai ancora creato un Network Load Balancer, scegli Crea nuovo sistema di bilanciamento del carico. Si apre la console Amazon EC2. Segui i passaggi descritti nella sezione Creazione di un Network Load Balancer a partire dal passaggio 3, quindi continua con questi passaggi nella console Amazon VPC.
5. Per le zone di disponibilità incluse `us-west-2aus-west-2b`, verificalo e `us-west-2c` appari nell'elenco.
6. Se non desideri accettare o rifiutare manualmente ogni richiesta di connessione inviata al servizio endpoint, in Impostazioni aggiuntive seleziona Accettazione richiesta. Se si deseleziona questa casella di controllo, il servizio di endpoint accetta automaticamente ogni richiesta di connessione che riceve.
7. Scegli Create (Crea).
8. Nel nuovo servizio endpoint, scegli Allow principal.
9. Contattaci per ottenere l'ARN dell'endpoint VPC Device Farm (servizio ARN) da aggiungere all'elenco degli indirizzi consentiti per il servizio endpoint, quindi aggiungi l'ARN del servizio all'elenco degli indirizzi consentiti per il servizio.
10. Nella scheda Details (Dettagli) per il servizio di endpoint, annotare il nome del servizio (service name (nome servizio)). Questo nome è necessario quando si crea la configurazione dell’endpoint VPC nella fase successiva.

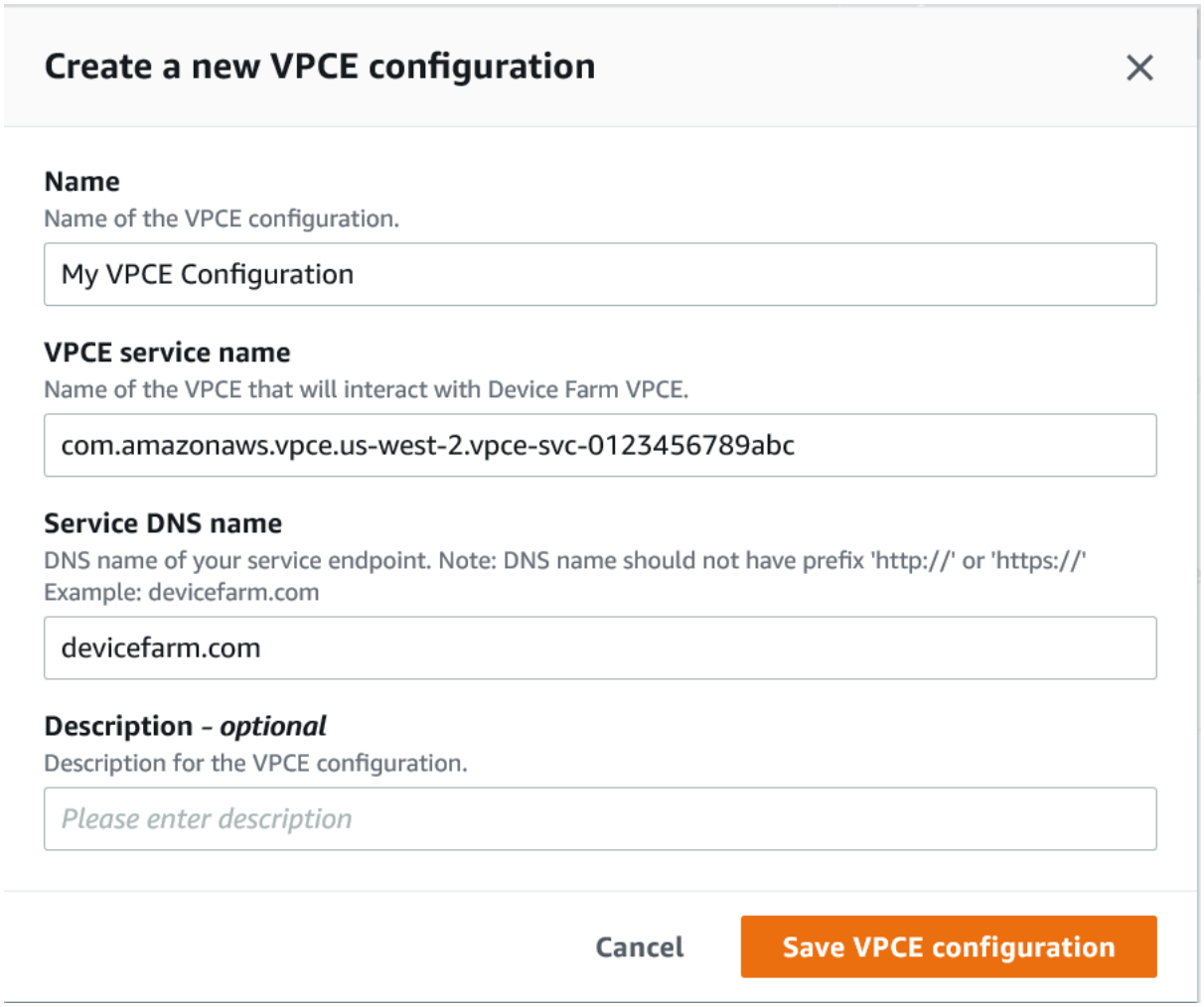
Il tuo servizio endpoint VPC è ora pronto per l'uso con Device Farm.

Fase 3: Creazione di una configurazione di endpoint VPC in Device Farm

Dopo aver creato un servizio endpoint in Amazon VPC, puoi creare una configurazione di endpoint Amazon VPC in Device Farm.

1. Accedere alla console Device Farm all'indirizzo https://console.aws.amazon.com/devicefarm.
2. Nel riquadro di navigazione, scegli Test per dispositivi mobili, quindi Dispositivi privati.
3. Scegli le configurazioni VPCE.
4. Scegli Crea configurazione VPCE.
5. In Crea una nuova configurazione VPCE, inserisci un nome per la configurazione dell'endpoint VPC.
6. Per il nome del servizio VPCE, inserisci il nome del servizio endpoint Amazon VPC (nome del servizio) che hai annotato nella console Amazon VPC. Il nome è simile a `com.amazonaws.vpce.us-west-2.vpce-svc-id`.
7. Per il nome DNS del servizio, inserisci il nome DNS del servizio per l'app che desideri testare (ad esempio,). `devicefarm.com` Non specificare `http` o `https` prima del nome DSN del servizio.

   Il nome di dominio non è accessibile tramite l'Internet pubblico. Inoltre, questo nuovo nome di dominio, mappato al tuo servizio endpoint VPC, è generato da Amazon Route 53 ed è disponibile esclusivamente per te nella tua sessione Device Farm.
8. Scegli Save (Salva).



Fase 4: Creazione di un test

Dopo aver salvato la configurazione dell'endpoint VPC, puoi utilizzare la configurazione per creare esecuzioni di test o sessioni di accesso remoto. Per ulteriori informazioni, consulta Creazione di un'esecuzione di test in Device Farm o Creazione di una sessione.

Registrazione delle chiamate API AWS Device Farm con AWS CloudTrail

AWS Device Farm è integrato con AWS CloudTrail un servizio che fornisce un registro delle azioni intraprese da un utente, un ruolo o un AWS servizio in AWS Device Farm. CloudTrail acquisisce tutte le chiamate API per AWS Device Farm come eventi. Le chiamate acquisite includono chiamate dalla console AWS Device Farm e chiamate in codice alle operazioni dell'API AWS Device Farm. Se crei un trail, puoi abilitare la distribuzione continua di CloudTrail eventi a un bucket Amazon S3, inclusi gli eventi per AWS Device Farm. Se non configuri un percorso, puoi comunque visualizzare gli eventi più recenti nella CloudTrail console nella cronologia degli eventi. Utilizzando le informazioni raccolte da CloudTrail, è possibile determinare la richiesta effettuata ad AWS Device Farm, l'indirizzo IP da cui è stata effettuata, chi ha effettuato la richiesta, quando è stata effettuata e ulteriori dettagli.

Per ulteriori informazioni CloudTrail, consulta la Guida AWS CloudTrail per l'utente.

Informazioni su AWS Device Farm in CloudTrail

CloudTrail è abilitato sul tuo AWS account al momento della creazione dell'account. Quando si verifica un'attività in AWS Device Farm, tale attività viene registrata in un CloudTrail evento insieme ad altri eventi di AWS servizio nella cronologia degli eventi. Puoi visualizzare, cercare e scaricare eventi recenti nel tuo AWS account. Per ulteriori informazioni, consulta Visualizzazione degli eventi con la cronologia degli CloudTrail eventi.

Per una registrazione continua degli eventi nel tuo AWS account, inclusi gli eventi per AWS Device Farm, crea un percorso. Un trail consente di CloudTrail inviare file di log a un bucket Amazon S3. Per impostazione predefinita, quando si crea un trail nella console, il trail sarà valido in tutte le regioni AWS. Il trail registra gli eventi di tutte le regioni della AWS partizione e consegna i file di log al bucket Amazon S3 specificato. Inoltre, puoi configurare altri AWS servizi per analizzare ulteriormente e agire in base ai dati sugli eventi raccolti nei log. CloudTrail Per ulteriori informazioni, consulta gli argomenti seguenti:

- Panoramica della creazione di un trail
- CloudTrail Servizi e integrazioni supportati
- Configurazione delle notifiche Amazon SNS per CloudTrail
- Ricezione di file di CloudTrail registro da più regioni e ricezione di file di CloudTrail registro da più account

Quando CloudTrail la registrazione è abilitata nell' AWS account, le chiamate API effettuate alle azioni di Device Farm vengono registrate nei file di registro. I record di Device Farm vengono scritti insieme ad altri record di AWS servizio in un file di registro. CloudTrail determina quando creare e scrivere su un nuovo file in base a un periodo di tempo e alle dimensioni del file.

Tutte le azioni di Device Farm vengono registrate e documentate nel AWS CLI riferimento e nel. Automazione di Device Farm Ad esempio, le chiamate per creare un nuovo progetto o per l'esecuzione in Device Farm generano voci nei file di CloudTrail registro.

Ogni evento o voce di log contiene informazioni sull'utente che ha generato la richiesta. Le informazioni di identità consentono di determinare quanto segue:

- Se la richiesta è stata effettuata con credenziali utente root o AWS Identity and Access Management (IAM).
- Se la richiesta è stata effettuata con le credenziali di sicurezza temporanee per un ruolo o un utente federato.
- Se la richiesta è stata effettuata da un altro AWS servizio.

Per ulteriori informazioni, consulta Elemento CloudTrail userIdentity.

Informazioni sulle voci dei file di log di AWS Device Farm

Un trail è una configurazione che consente la distribuzione di eventi come file di log in un bucket Amazon S3 specificato dall'utente. CloudTrail i file di registro contengono una o più voci di registro. Un evento rappresenta una singola richiesta proveniente da qualsiasi fonte e include informazioni sull'azione richiesta, la data e l'ora dell'azione, i parametri della richiesta e così via. CloudTrail i file di registro non sono una traccia ordinata dello stack delle chiamate API pubbliche, quindi non vengono visualizzati in un ordine specifico.

L'esempio seguente mostra una voce di CloudTrail registro che dimostra l'`ListRuns`azione Device Farm:

```
{
  "Records": [
    {
      "eventVersion": "1.03",
      "userIdentity": {
        "type": "Root",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
```

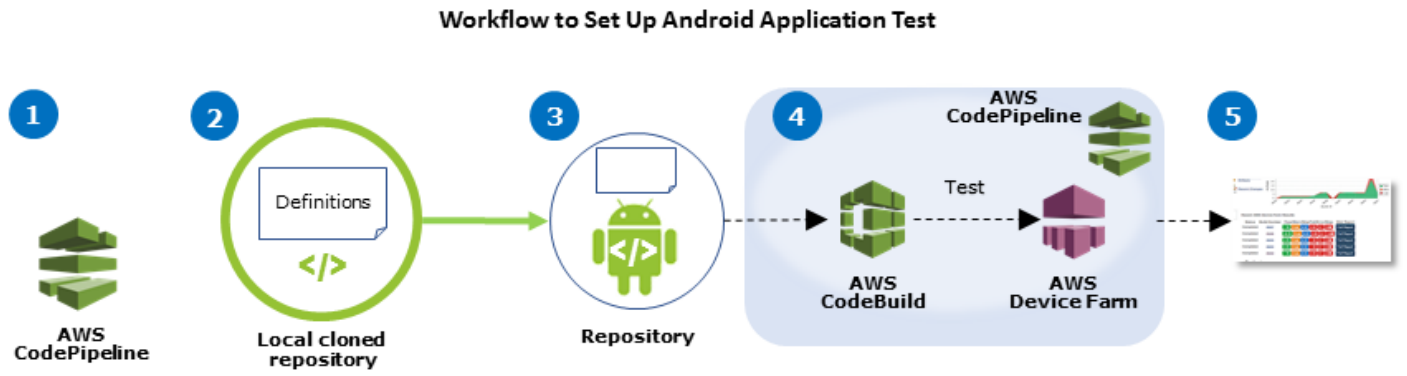
```
        "arn": "arn:aws:iam::123456789012:root",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2015-07-08T21:13:35Z"
          }
        }
      },
      "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 ...
          ]
```

```
          }
        }
      }
    ]
}
```

Integrazione di AWS Device Farm in una fase di CodePipeline test

Puoi utilizzarlo AWS CodePipelineper incorporare i test delle app mobili configurati in Device Farm in una pipeline di rilascio automatizzata gestita da AWS. Puoi configurare la tua pipeline in modo che esegua test su richiesta, sulla base di un programma o come parte di un flusso di integrazione continua.

Il seguente diagramma mostra il flusso di integrazione continua in cui un'app Android viene compilata e testata ogni volta che viene eseguito il push di un commit nel suo archivio. Per creare questa configurazione di pipeline, consulta il Tutorial: Crea e testa un'app Android quando viene inviata. GitHub



1. Configura	2. Aggiungi definizioni	3. Push	4. Costruisci e testa	5. Report
Configura le risorse della pipeline	Aggiungi al pacchetto la compilazione e le definizioni di test	Invia un pacchetto al repository	Compilazione dell'app e test di artefatti di output della compilazione avviati automaticamente	Visualizza i risultati del test

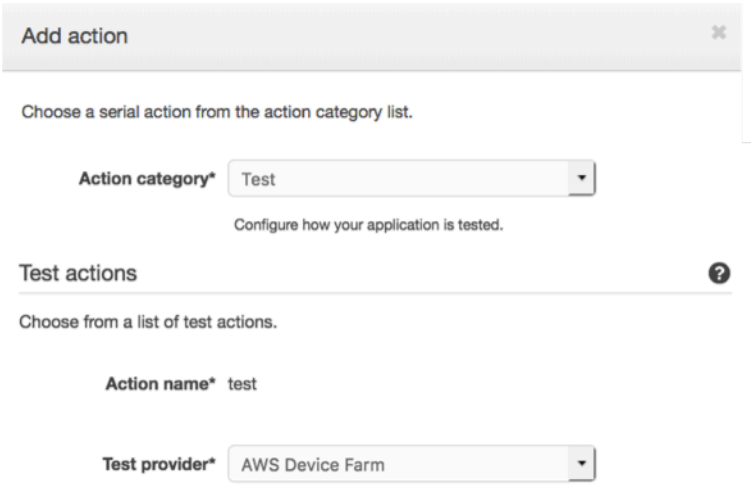
Per imparare a configurare una pipeline che testi in modo continuo un'app compilata (come un file `.ipa` per iOS o `.apk` per Android) come sua origine, consulta l'Esercitazione: testa un'app iOS ogni volta che carichi un file .ipa su un bucket Amazon S3.

Configura CodePipeline per utilizzare i test Device Farm

In questi passaggi, si presuppone che tu abbia configurato un progetto Device Farm e creato una pipeline. La pipeline deve essere configurata con una fase di test che riceva un artefatto di input contenente la definizione del tuo test e i file del pacchetto dell'app compilata. L'artefatto di input della fase di test può essere l'artefatto di output di una fase di origine o di compilazione configurata nella tua pipeline.

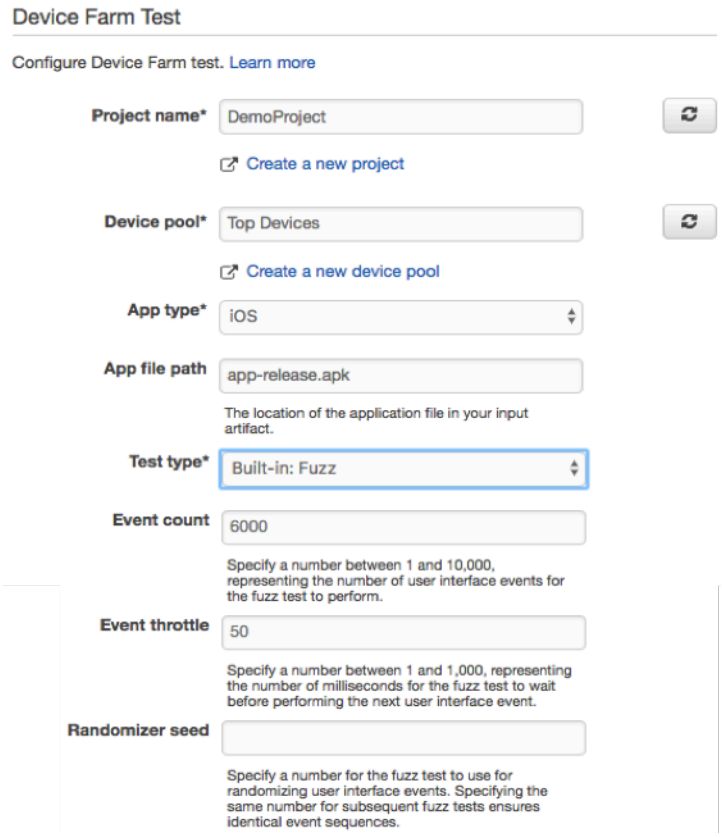
Per configurare un test di Device Farm, eseguire un CodePipeline test come azione di test

1. Accedi a Console di gestione AWS e apri la CodePipeline console all'indirizzo https://console.aws.amazon.com/codepipeline/.
2. Scegliere la pipeline per la versione della propria app.
3. Nel pannello della fase di test, scegliere l'icona a forma di matita e selezionare quindi Action (Azione).
4. Nel pannello Add action (Aggiungi azione), per Action category (Categoria dell'azione), selezionare Test.
5. Alla voce Action name (Nome operazione), inserire un nome.
6. Alla voce Test provider (Provider del test), scegliere AWS Device Farm.



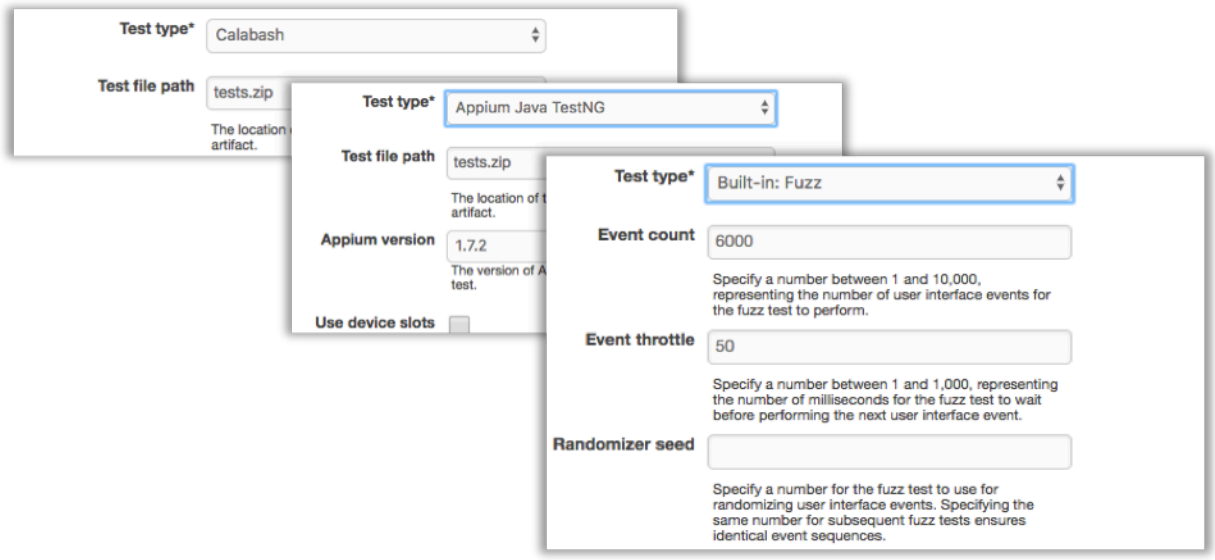
7. In Nome progetto, scegli il tuo progetto Device Farm esistente o scegli Crea un nuovo progetto.

8. Alla voce Device pool (Pool di dispositivi), scegliere il pool di dispositivi preesistenti oppure scegliere Create a new device pool (Crea un nuovo pool di dispositivi). Se crei un pool di dispositivi, devi selezionare un set di dispositivi di test.
9. In App type (Tipo di app), selezionare la piattaforma della propria app.

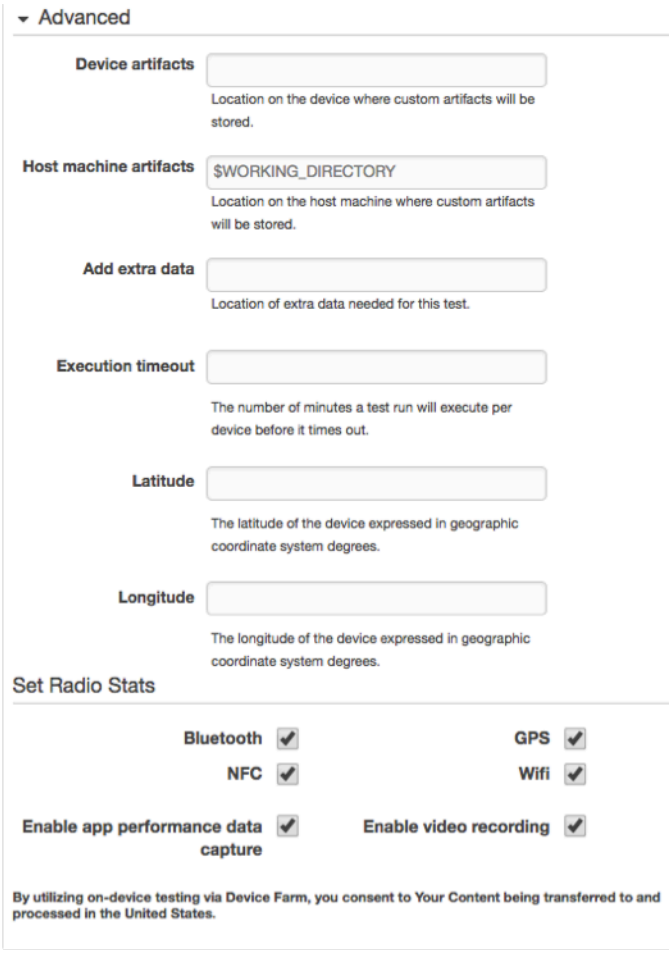


10. Alla voce App file path (Percorso file app), inserire il percorso del pacchetto dell'applicazione compilata. Il percorso è relativo alla cartella principale dell'artefatto di input del test.
11. Alla voce Test type (Tipo di test), procedere in uno dei seguenti modi:

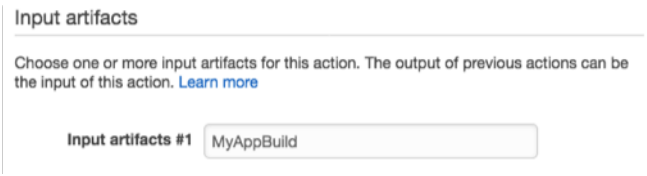
    - Se utilizzi uno dei test Device Farm integrati, scegli il tipo di test configurato nel tuo progetto Device Farm.
    - Se non stai utilizzando uno dei test integrati di Device Farm, nel percorso del file di test, inserisci il percorso del file di definizione del test. Il percorso è relativo alla cartella principale dell'artefatto di input del test.



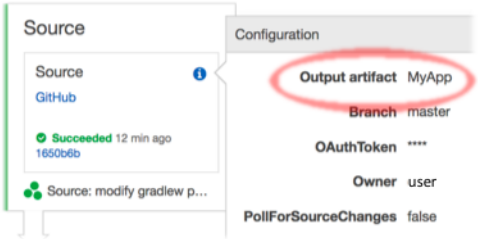
12. Nei campi rimanenti, inserire la configurazione appropriata per il test e il tipo di applicazione.
13. (Facoltativo) In Advanced (Avanzate), fornire informazioni di configurazione dettagliate per la sessione di test.



14. Alla voce Input artifact (Artefatti di input), scegliere l'artefatto di input che corrisponde all'artefatto di output della fase che precede la fase di test nella pipeline.



Nella CodePipeline console, puoi trovare il nome dell'elemento di output per ogni fase passando il mouse sull'icona delle informazioni nel diagramma della pipeline. Se la pipeline verifica l'app direttamente dalla fase Source, scegli. MyApp Se la tua pipeline include una fase di creazione, scegli. MyAppBuild



15. Nella parte inferiore del pannello, scegliere Add Action (Aggiungi operazione).
16. Nel CodePipeline riquadro, scegli Salva modifica alla pipeline, quindi scegli Salva modifica.
17. Per inviare le modifiche e avviare una compilazione tramite pipeline, scegliere Release change (Rilascia modifica) e quindi scegliere Release (Rilascia).

AWS CLI riferimento per AWS Device Farm

Per utilizzare AWS Command Line Interface (AWS CLI) per eseguire i comandi Device Farm, consulta la pagina di AWS CLI riferimento per AWS Device Farm.

Per informazioni generali su AWS CLI, consulta la Guida per l'AWS Command Line Interface utente e il AWS CLI Command Reference.

PowerShell Riferimento Windows per AWS Device Farm

Per utilizzare Windows per PowerShell eseguire i comandi di Device Farm, vedere il riferimento al cmdlet Device Farm nella Guida di riferimento ai AWS Tools for Windows PowerShell cmdlet. Per ulteriori informazioni, consulta Configurazione degli strumenti AWS per Windows PowerShell nella Guida per l'AWS Strumenti per PowerShell utente.

Automazione di AWS Device Farm

L'accesso programmatico a Device Farm è un modo efficace per automatizzare le attività più comuni da svolgere, come la pianificazione di un'esecuzione o il download degli artefatti per un'esecuzione, una suite o un test. L' AWS SDK e provide offrono i mezzi per farlo. AWS CLI

L' AWS SDK fornisce l'accesso a tutti i AWS servizi, inclusi Device Farm, Amazon S3 e altro ancora. Per ulteriori informazioni, consultare la pagina

- gli AWS strumenti e SDKs
- riferimento all'API AWS Device Farm

Esempio: utilizzo della AWS CLI o dell'SDK per caricare un'app o un test su Device Farm

Gli esempi seguenti mostrano come creare un caricamento su Device Farm utilizzando la AWS CLI o utilizzando l' AWS SDK in varie lingue. I caricamenti sono gli elementi fondamentali per la pianificazione delle esecuzioni di test su Device Farm e includono quanto segue:

- La tua app
- Il tuo test
- Il tuo file delle specifiche del test

I caricamenti vengono creati utilizzando l'`CreateUpload`API. Questa API restituisce un URL predefinito S3 a cui puoi inviare il caricamento utilizzando una richiesta HTTP PUT. L'URL scade dopo 24 ore.

AWS CLI

Nota: questo esempio utilizza lo strumento da riga di comando *`curl`* per inviare l'app a Device Farm.

Innanzitutto, crea un progetto se non l'hai già fatto.

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

Questo mostrerà un output come il seguente:

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

Quindi, procedi come segue per creare il caricamento e inviarlo a Device Farm. In questo esempio, creeremo il caricamento di un'app Android utilizzando un file APK locale. Per ulteriori informazioni sui tipi di caricamento, inclusi i dettagli sui tipi di caricamento delle app iOS, consulta la nostra documentazione API per la creazione di un `Upload`.

```
$ export APP_PATH="/local/path/to/my_sample_app.apk"
$ export APP_TYPE="ANDROID_APP"
```

Per prima cosa, creiamo il caricamento 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"
```

Questo mostrerà un output come il seguente:

```
{
    "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"
    }
}
```

Quindi, esegui una chiamata PUT usando curl per inviare l'app al bucket S3 di Device Farm:

```
$ curl -T "$APP_PATH" "https://prod-us-west-2-uploads.s3.dualstack.us-
west-2.amazonaws.com/arn%3Aaws%3Adevicefarm%3Aus-west-2..."
```

Infine, attendi che l'app abbia lo stato «riuscito»:

```
$ aws devicefarm get-upload --arn "arn:aws:devicefarm:us-
west-2:385076942068:upload:490a6350-0ba3-43e5-83f5-d2896b069a34/a120e848-c57b-4e8d-
a720-d750a0c4d936"
```

Questo mostrerà un output come il seguente:

```
{
    "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

Nota: questo esempio utilizza il *`requests`* pacchetto di terze parti per inviare l'app a Device Farm, oltre all' AWS SDK per Python*`boto3`*.

Innanzitutto, crea un progetto se non l'hai già fatto.

```
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
#     }
# }
```

Quindi, procedi come segue per creare il caricamento e inviarlo a Device Farm. In questo esempio, creeremo il caricamento di un'app Android utilizzando un file APK locale. Per ulteriori informazioni sui tipi di caricamento, inclusi i dettagli sui tipi di caricamento delle app iOS, consulta la nostra documentazione API per la creazione di un `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

Nota: questo esempio utilizza l' AWS SDK for Java v2 *`HttpClient`* e per inviare l'app a Device Farm ed è compatibile con le versioni JDK 11 e successive.

Innanzitutto, crea un progetto se non l'hai già fatto.

```
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=...}
}
```

Quindi, procedi come segue per creare il caricamento e inviarlo a Device Farm. In questo esempio, creeremo il caricamento di un'app Android utilizzando un file APK locale. Per ulteriori informazioni sui tipi di caricamento, inclusi i dettagli sui tipi di caricamento delle app iOS, consulta la nostra documentazione API per la creazione di un `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

Nota: questo esempio utilizza AWS SDK for JavaScript (v3) e Node 18+ per inviare l'app *fetch* a Device Farm.

Innanzitutto, crea un progetto se non l'hai già fatto.

```
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 } }
```

Quindi, procedi come segue per creare il caricamento e inviarlo a Device Farm. In questo esempio, creeremo il caricamento di un'app Android utilizzando un file APK locale. Per ulteriori informazioni sui tipi di caricamento, inclusi i dettagli sui tipi di caricamento delle app iOS, consulta la nostra documentazione API per la creazione di un `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#

Nota: questo esempio utilizza l' AWS SDK for .NET *`HttpClient`* e invia l'app a Device Farm.

Innanzitutto, crea un progetto se non l'hai già fatto.

```
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 = ... }
```

Quindi, procedi come segue per creare il caricamento e inviarlo a Device Farm. In questo esempio, creeremo il caricamento di un'app Android utilizzando un file APK locale. Per ulteriori informazioni sui tipi di caricamento, inclusi i dettagli sui tipi di caricamento delle app iOS, consulta la nostra documentazione API per la creazione di un `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

Nota: questo esempio utilizza l' AWS SDK for *Net::HTTP* Ruby e per inviare l'app a Device Farm.

Innanzitutto, crea un progetto se non l'hai già fatto.

```
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>
```

Quindi, procedi come segue per creare il caricamento e inviarlo a Device Farm. In questo esempio, creeremo il caricamento di un'app Android utilizzando un file APK locale. Per ulteriori informazioni sui tipi di caricamento, inclusi i dettagli sui tipi di caricamento delle app iOS, consulta la nostra documentazione API per la creazione di un `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
```

Esempio: utilizzo dell' AWS SDK per avviare un'esecuzione di Device Farm e raccogliere artefatti

L'esempio seguente fornisce una beginning-to-end dimostrazione di come utilizzare l' AWS SDK per lavorare con Device Farm. Inoltre, vengono effettuate le seguenti operazioni:

- Carica un test e pacchetti applicativi su Device Farm
- Avvia un'esecuzione di test e attende il suo completamento (o errore)
- Scarica tutti gli artefatti prodotti dalle suite di test

Questo esempio dipende dal pacchetto `requests` di terze parti per interagire con HTTP.

```
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")
```

Risoluzione degli errori di Device Farm

In questa sezione, troverai messaggi di errore e procedure per aiutarti a risolvere i problemi più comuni con Device Farm.

> **Note**
>
> Per risolvere i test di Appium che falliscono inaspettatamente su Device Farm, consulta la nostra guida per i test di Appium lato client

Argomenti

Risoluzione dei problemi relativi ai test delle applicazioni Android in AWS Device Farm

Il seguente argomento elenca i messaggi di errore che si verificano durante il caricamento dei test di applicazioni Android e consiglia soluzioni alternative per risolvere ogni errore.

> **Note**
>
> Le seguenti istruzioni si basano su Linux x86_64 e Mac.

ANDROID_APP_UNZIP_FAILED

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile aprire l'applicazione. Verificare che il file sia valido e riprovare.

Verificare che sia possibile decomprimere il pacchetto dell'applicazione senza errori. Nel seguente esempio, il nome del pacchetto è app-debug.apk.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip app-debug.apk
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Un pacchetto di un'applicazione Android valido dovrebbe produrre un output come il seguente:

   ```
   .
   |-- AndroidManifest.xml
   |-- classes.dex
   |-- resources.arsc
   |-- assets (directory)
   |-- res (directory)
   `-- META-INF (directory)
   ```

   Per ulteriori informazioni, consulta Test Android in AWS Device Farm.

ANDROID_APP_AAPT_DEBUG_BADGING_FAILED

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile estrarre informazioni sull'applicazione. Verificare che l'applicazione sia valida eseguendo il comando `aapt debug badging` *`<path to your test package>`* e riprovare quando il comando non stampa più errori.

Durante il processo di convalida del caricamento, AWS Device Farm analizza le informazioni dall'output di un `aapt debug badging` *`<path to your package>`* comando.

Controllare che questo comando possa essere eseguito correttamente sull'applicazione Android. Nel seguente esempio, il nome del pacchetto è app-debug.apk.

- Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando:

  ```
  $ aapt debug badging app-debug.apk
  ```

  Un pacchetto di un'applicazione Android valido dovrebbe produrre un output come il seguente:

  ```
  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'
  ```

  Per ulteriori informazioni, consulta Test Android in AWS Device Farm.

ANDROID_APP_PACKAGE_NAME_VALUE_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare il valore del nome del pacchetto nell'applicazione. Verificare che l'applicazione sia valida eseguendo il comando `aapt debug badging` *`<path to your test package>`* e riprovare dopo la ricerca del valore del nome del pacchetto con parola chiave "pacchetto: nome".

Durante il processo di convalida del caricamento, AWS Device Farm analizza il valore del nome del pacchetto dall'output di un `aapt debug badging` *`<path to your package>`* comando.

Controllare che questo comando possa essere eseguito correttamente sull'applicazione Android e che trovi con successo il valore del nome del pacchetto. Nel seguente esempio, il nome del pacchetto è app-debug.apk.

- Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

  ```
  $ aapt debug badging app-debug.apk | grep "package: name="
  ```

  Un pacchetto di un'applicazione Android valido dovrebbe produrre un output come il seguente:

  ```
  package: name='com.amazon.aws.adf.android.referenceapp' versionCode='1'
   versionName='1.0' platformBuildVersionName='5.1.1-1819727'
  ```

  Per ulteriori informazioni, consulta Test Android in AWS Device Farm.

ANDROID_APP_SDK_VERSION_VALUE_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare il valore della versione SDK nell'applicazione. Verificare che l'applicazione sia valida eseguendo il comando `aapt debug badging` *`<path to your test`*

package> e riprovare dopo la ricerca del valore della versione SDK con parola chiave `sdkVersion`.

Durante il processo di convalida del caricamento, AWS Device Farm analizza il valore della versione dell'SDK dall'output di un comando. `aapt debug badging` *<path to your package>*

Controllare che questo comando possa essere eseguito correttamente sull'applicazione Android e che trovi con successo il valore del nome del pacchetto. Nel seguente esempio, il nome del pacchetto è app-debug.apk.

- Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

  ```
  $ aapt debug badging app-debug.apk | grep "sdkVersion"
  ```

  Un pacchetto di un'applicazione Android valido dovrebbe produrre un output come il seguente:

  ```
  sdkVersion:'9'
  ```

  Per ulteriori informazioni, consulta Test Android in AWS Device Farm.

ANDROID_APP_AAPT_DUMP_XMLTREE_FAILED

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Non siamo riusciti a trovare il AndroidManifest file.xml valido nella tua applicazione. Verificare che il pacchetto di test sia valido eseguendo il comando `aapt dump xmltree` *<path to your test package>* `AndroidManifest.xml` e riprovare quando il comando non stampa più errori.

Durante il processo di convalida del caricamento, AWS Device Farm analizza le informazioni dall'albero di analisi XML per un file XML contenuto nel pacchetto utilizzando il comando. `aapt dump xmltree` *<path to your package>* `AndroidManifest.xml`

Controllare che questo comando possa essere eseguito correttamente sull'applicazione Android. Nel seguente esempio, il nome del pacchetto è app-debug.apk.

- Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

```
$ aapt dump xmltree app-debug.apk. AndroidManifest.xml
```

Un pacchetto di un'applicazione Android valido dovrebbe produrre un output come il seguente:

```
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")
```

Per ulteriori informazioni, consulta Test Android in AWS Device Farm.

ANDROID_APP_DEVICE_ADMIN_PERMISSIONS

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> L'applicazione richiede autorizzazioni di amministratore del dispositivo. Verificare che le autorizzazioni non siano richieste eseguendo il comando `aapt dump xmltree <path to`

your test package> AndroidManifest.xml e riprovare dopo aver controllato che l'output non contenga la parola chiave android.permission.BIND_DEVICE_ADMIN.

Durante il processo di convalida del caricamento, AWS Device Farm analizza le informazioni di autorizzazione dall'albero di analisi xml per un file xml contenuto nel pacchetto utilizzando il comando. aapt dump xmltree *<path to your package>* AndroidManifest.xml

Verificare che l'applicazione non richieda autorizzazioni di amministratore per il dispositivo. Nel seguente esempio, il nome del pacchetto è app-debug.apk.

- Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

```
$ aapt dump xmltree app-debug.apk AndroidManifest.xml
```

L'output dovrebbe essere come segue:

```
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")
        ……
```

Se l'applicazione Android è valida, l'output non deve contenere: A: android:name(0x01010003)="android.permission.BIND_DEVICE_ADMIN" (Raw: "android.permission.BIND_DEVICE_ADMIN").

Per ulteriori informazioni, consulta Test Android in AWS Device Farm.

Alcune finestre della mia applicazione Android mostrano una schermata vuota o nera

Se state testando un'applicazione Android e notate che alcune finestre dell'applicazione appaiono con una schermata nera nella registrazione video del test effettuata da Device Farm, è possibile che l'applicazione utilizzi la `FLAG_SECURE` funzionalità di Android. Questo flag (come descritto nella documentazione ufficiale di Android) viene utilizzato per impedire che determinate finestre di un'applicazione vengano registrate dagli strumenti di registrazione dello schermo. Di conseguenza, la funzione di registrazione dello schermo di Device Farm (sia per l'automazione che per i test di accesso remoto) potrebbe mostrare una schermata nera al posto della finestra dell'applicazione se la finestra utilizza questo flag.

Questo flag viene spesso utilizzato dagli sviluppatori per le pagine delle loro applicazioni che contengono informazioni sensibili come le pagine di accesso. Se vedi una schermata nera al posto della schermata dell'applicazione in alcune pagine come la pagina di accesso, collabora con i tuoi sviluppatori per ottenere una versione dell'applicazione che non utilizzi questo flag per i test.

Inoltre, tenete presente che Device Farm può ancora interagire con le finestre delle applicazioni che hanno questo flag. Pertanto, se la pagina di accesso dell'applicazione appare come una schermata nera, è comunque possibile inserire le credenziali per accedere all'applicazione (e quindi visualizzare le pagine non bloccate dal `FLAG_SECURE` flag).

Risoluzione dei problemi dei JUnit test Java di Appium in AWS Device Farm

L'argomento seguente elenca i messaggi di errore che si verificano durante il caricamento dei JUnit test Java di Appium e consiglia soluzioni alternative per risolvere ogni errore.

> **Note**
>
> Le seguenti istruzioni si basano su Linux x86_64 e Mac.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile aprire il file ZIP del test. Verificare che il file sia valido e riprovare.

Verificare che sia possibile decomprimere il pacchetto di test senza errori. Nell'esempio seguente, il nome del pacchetto è .zip. zip-with-dependencies

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Un JUnit pacchetto Appium Java valido dovrebbe produrre un output simile al seguente:

   ```
   .
   |— 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
   ```

   Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_DEPENDENCY_DIR_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare la directory dependency-jars nel pacchetto di test. Decomprimere il pacchetto di test, verificare che la directory dependency-jars sia nel pacchetto e riprovare.

Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies .zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il JUnit pacchetto Appium Java è valido, troverai la *`dependency-jars`* directory all'interno della directory di lavoro:

   ```
   .
   |— 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
   ```

   Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DI

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare un file JAR nella struttura ad albero delle directory dependency-jars. Decomprimere il pacchetto di test, quindi aprire la directory dependency-jars, verificare che almeno un file JAR si trovi nella directory e riprovare.

Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies .zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il JUnit pacchetto Appium Java è valido, troverai almeno un *jar* file all'interno della directory: *dependency-jars*

   ```
   .
   |— 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
   ```

   Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare un file *-tests.jar nel pacchetto di test. Decomprimere il pacchetto di test, verificare che nel pacchetto sia presente almeno un file *-tests.jar e riprovare.

Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies .zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il JUnit pacchetto Appium Java è valido, troverai almeno un *`jar`* file come *`acme-android-appium-1.0-SNAPSHOT-tests.jar`* nel nostro esempio. Il nome del file può essere diverso, ma dovrebbe terminare con. *`-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
   ```

   Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JA

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare un file di classe all'interno del file JAR dei test. Decomprimere il pacchetto di test, quindi decomprimere il file JAR dei test e verificare che almeno un file di classe si trovi nel file JAR, quindi riprovare.

Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Dovresti trovare almeno un file jar come *acme-android-appium-1.0-SNAPSHOT-tests.jar* nel nostro esempio. Il nome del file può essere diverso, ma dovrebbe terminare con*-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. Dopo aver estratto correttamente i file, si dovrebbe trovare almeno una classe nella struttura delle directory di lavoro, eseguendo il comando:

```
$ tree .
```

L'output visualizzato dovrebbe essere di questo tipo:

```
.
|— 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
```

Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_JUNIT_VERSION_VALUE_UNKNOWN

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Non siamo riusciti a trovare un valore di JUnit versione. Decomprimi il pacchetto di test e apri la directory dependency-jars, verifica che il file JUnit JAR sia all'interno della directory e riprova.

Nell'esempio seguente, il nome del pacchetto è .zip. zip-with-dependencies

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip zip-with-dependencies.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
tree .
```

L'output dovrebbe essere simile al seguente:

```
.
|— 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
```

Se il JUnit pacchetto Appium Java è valido, nel nostro esempio troverai il file delle JUnit dipendenze simile al file *junit-4.10.jar* jar. Il nome deve essere composto dalla parola chiave *junit* e dal relativo numero di versione, che in questo esempio è 4.10.

Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Abbiamo riscontrato che la JUnit versione era inferiore alla versione minima 4.10 supportata. Cambia la JUnit versione e riprova.

Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip zip-with-dependencies.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare un file di JUnit dipendenza come *junit-4.10.jar* nel nostro esempio e il relativo numero di versione, che nel nostro esempio è 4.10:

```
.
|— 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**
>
> I test potrebbero non essere eseguiti correttamente se la JUnit versione specificata nel pacchetto di test è inferiore alla versione minima 4.10 supportata.

Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

Risoluzione dei problemi dei test delle applicazioni JUnit web Appium Java in AWS Device Farm

Il seguente argomento elenca i messaggi di errore che si verificano durante il caricamento dei test dell'applicazione JUnit Web Appium Java e consiglia soluzioni alternative per risolvere ogni errore. Per ulteriori informazioni sull'utilizzo di Appium con Device Farm, vedere. the section called “Test Appium automatici”

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile aprire il file ZIP del test. Verificare che il file sia valido e riprovare.

Verificare che sia possibile decomprimere il pacchetto di test senza errori. Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies .zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Un JUnit pacchetto Appium Java valido dovrebbe produrre un output simile al seguente:

   ```
   .
   |— 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

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare la directory dependency-jars nel pacchetto di test. Decomprimere il pacchetto di test, verificare che la directory dependency-jars sia nel pacchetto e riprovare.

Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies .zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il JUnit pacchetto Appium Java è valido, troverai la *`dependency-jars`* directory all'interno della directory di lavoro:

   ```
   .
   |— 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

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare un file JAR nella struttura ad albero delle directory dependency-jars. Decomprimere il pacchetto di test, quindi aprire la directory dependency-jars, verificare che almeno un file JAR si trovi nella directory e riprovare.

Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies .zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il JUnit pacchetto Appium Java è valido, troverai almeno un *jar* file all'interno della directory: *dependency-jars*

   ```
   .
   |– 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

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare un file *-tests.jar nel pacchetto di test. Decomprimere il pacchetto di test, verificare che nel pacchetto sia presente almeno un file *-tests.jar e riprovare.

Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies .zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il JUnit pacchetto Appium Java è valido, troverai almeno un *jar* file come *acme-android-appium-1.0-SNAPSHOT-tests.jar* nel nostro esempio. Il nome del file può essere diverso, ma dovrebbe terminare con. *-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
   ```

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TES

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare un file di classe all'interno del file JAR dei test. Decomprimere il pacchetto di test, quindi decomprimere il file JAR dei test e verificare che almeno un file di classe si trovi nel file JAR, quindi riprovare.

Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Dovresti trovare almeno un file jar come *acme-android-appium-1.0-SNAPSHOT-tests.jar* nel nostro esempio. Il nome del file può essere diverso, ma dovrebbe terminare con*-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. Dopo aver estratto correttamente i file, si dovrebbe trovare almeno una classe nella struttura delle directory di lavoro, eseguendo il comando:

```
$ tree .
```

L'output visualizzato dovrebbe essere di questo tipo:

```
.
|— 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

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Non siamo riusciti a trovare un valore di JUnit versione. Decomprimi il pacchetto di test e apri la directory dependency-jars, verifica che il file JUnit JAR sia all'interno della directory e riprova.

Nell'esempio seguente, il nome del pacchetto è .zip. zip-with-dependencies

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip zip-with-dependencies.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
tree .
```

L'output dovrebbe essere simile al seguente:

```
.
|— 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
```

Se il JUnit pacchetto Appium Java è valido, nel nostro esempio troverai il file delle JUnit dipendenze simile al file *junit-4.10.jar* jar. Il nome deve essere composto dalla parola chiave *junit* e dal relativo numero di versione, che in questo esempio è 4.10.

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Abbiamo riscontrato che la JUnit versione era inferiore alla versione minima 4.10 supportata. Cambia la JUnit versione e riprova.

Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip zip-with-dependencies.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare un file di JUnit dipendenza come *junit-4.10.jar* nel nostro esempio e il relativo numero di versione, che nel nostro esempio è 4.10:

```
.
|— 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**
>
> I test potrebbero non essere eseguiti correttamente se la JUnit versione specificata nel pacchetto di test è inferiore alla versione minima 4.10 supportata.

Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

Risoluzione dei problemi dei test Appium Java TestNg in AWS Device Farm

Il seguente argomento elenca i messaggi di errore che si verificano durante il caricamento dei test di Appium Java TestNG e consiglia soluzioni alternative per risolvere ogni errore.

> **Note**
>
> Le seguenti istruzioni si basano su Linux x86_64 e Mac.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile aprire il file ZIP del test. Verificare che il file sia valido e riprovare.

Verificare che sia possibile decomprimere il pacchetto di test senza errori. Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies .zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Un JUnit pacchetto Appium Java valido dovrebbe produrre un output simile al seguente:

   ```
   .
   |— 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
   ```

Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare la directory `dependency-jars` nel pacchetto di test. Decomprimere il pacchetto di test, verificare che la directory `dependency-jars` sia nel pacchetto e riprovare.

Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies .zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il JUnit pacchetto Appium Java è valido, troverai la *`dependency-jars`* directory all'interno della directory di lavoro.

   ```
   .
   |— 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
   ```

Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare un file JAR nella struttura ad albero delle directory dependency-jars. Decomprimere il pacchetto di test, quindi aprire la directory dependency-jars, verificare che almeno un file JAR si trovi nella directory e riprovare.

Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies .zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il JUnit pacchetto Appium Java è valido, troverai almeno un *jar* file all'interno della directory. *dependency-jars*

   ```
   .
   |— 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
   ```

Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare un file *-tests.jar nel pacchetto di test. Decomprimere il pacchetto di test, verificare che nel pacchetto sia presente almeno un file *-tests.jar e riprovare.

Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies .zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il JUnit pacchetto Appium Java è valido, troverai almeno un *jar* file come *acme-android-appium-1.0-SNAPSHOT-tests.jar* nel nostro esempio. Il nome del file può essere diverso, ma dovrebbe terminare con. *-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
   ```

Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare un file di classe all'interno del file JAR dei test. Decomprimere il pacchetto di test, quindi decomprimere il file JAR dei test e verificare che almeno un file di classe si trovi nel file JAR, quindi riprovare.

Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Dovresti trovare almeno un file jar come *acme-android-appium-1.0-SNAPSHOT-tests.jar* nel nostro esempio. Il nome del file può essere diverso, ma dovrebbe terminare con*-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. Per estrarre i file dal file jar, eseguire il seguente comando:

```
$ jar xf acme-android-appium-1.0-SNAPSHOT-tests.jar
```

4. Dopo aver estratto correttamente i file, eseguire il comando seguente:

```
$ tree .
```

Dovresti trovare almeno una classe nella struttura ad albero della directory di lavoro:

```
.
|– 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
```

Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

Risoluzione dei problemi delle applicazioni Web Appium Java TestNg in AWS Device Farm

Il seguente argomento elenca i messaggi di errore che si verificano durante il caricamento dei test per applicazioni Web di Appium Java TestNG e consiglia soluzioni alternative per risolvere ogni errore.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile aprire il file ZIP del test. Verificare che il file sia valido e riprovare.

Verificare che sia possibile decomprimere il pacchetto di test senza errori. Nell'esempio seguente, il nome del pacchetto è .zip. zip-with-dependencies

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Un JUnit pacchetto Appium Java valido dovrebbe produrre un output simile al seguente:

   ```
   .
   |— 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
   ```

   Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSIN

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare la directory dependency-jars nel pacchetto di test. Decomprimere il pacchetto di test, verificare che la directory dependency-jars sia nel pacchetto e riprovare.

Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies .zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il JUnit pacchetto Appium Java è valido, troverai la *dependency-jars* directory all'interno della directory di lavoro.

   ```
   .
   |— 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
   ```

   Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPEND

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare un file JAR nella struttura ad albero delle directory dependency-jars. Decomprimere il pacchetto di test, quindi aprire la directory dependency-jars, verificare che almeno un file JAR si trovi nella directory e riprovare.

Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies .zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il JUnit pacchetto Appium Java è valido, troverai almeno un *`jar`* file all'interno della directory. *`dependency-jars`*

   ```
   .
   |— 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
   ```

   Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare un file *-tests.jar nel pacchetto di test. Decomprimere il pacchetto di test, verificare che nel pacchetto sia presente almeno un file *-tests.jar e riprovare.

Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies .zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il JUnit pacchetto Appium Java è valido, troverai almeno un *jar* file come *acme-android-appium-1.0-SNAPSHOT-tests.jar* nel nostro esempio. Il nome del file può essere diverso, ma dovrebbe terminare con. *-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
   ```

   Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_T

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare un file di classe all'interno del file JAR dei test. Decomprimere il pacchetto di test, quindi decomprimere il file JAR dei test e verificare che almeno un file di classe si trovi nel file JAR, quindi riprovare.

Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip zip-with-dependencies.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

   Dovresti trovare almeno un file jar come *acme-android-appium-1.0-SNAPSHOT-tests.jar* nel nostro esempio. Il nome del file può essere diverso, ma dovrebbe terminare con*-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. Per estrarre i file dal file jar, eseguire il seguente comando:

```
$ jar xf acme-android-appium-1.0-SNAPSHOT-tests.jar
```

4. Dopo aver estratto correttamente i file, eseguire il comando seguente:

```
$ tree .
```

Dovresti trovare almeno una classe nella struttura ad albero della directory di lavoro:

```
.
|— 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
```

Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

Risoluzione dei problemi dei test Appium Python in AWS Device Farm

Il seguente argomento elenca i messaggi di errore che si verificano durante il caricamento dei test di Appium Python e consiglia soluzioni alternative per risolvere ogni errore.

APPIUM_PYTHON_TEST_PACKAGE_UNZIP_FAILED

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile aprire il file ZIP del test Appium. Verificare che il file sia valido e riprovare.

Verificare che sia possibile decomprimere il pacchetto di test senza errori. In questo esempio, il nome del pacchetto è test_bundle.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip test_bundle.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Un pacchetto Appium Python valido dovrebbe produrre un output come il seguente:

```
.
|-- 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
```

Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare un file wheel di dipendenza nella struttura delle directory wheelhouse. Decomprimere il pacchetto di test, quindi aprire la directory wheelhouse, verificare che nella directory si trovi almeno un file wheel e riprovare.

Verificare che sia possibile decomprimere il pacchetto di test senza errori. In questo esempio, il nome del pacchetto è test_bundle.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip test_bundle.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

   Se il pacchetto Appium Python è valido, troverai almeno un file *.whl* dipendente come i file evidenziati all'interno della directory. *wheelhouse*

```
.
|-- 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
```

   Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_INVALID_PLATFORM

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Trovato almeno un file wheel che specificava una piattaforma non supportata. Decomprimere il pacchetto di test, quindi aprire la directory wheelhouse, verificare che i nomi dei file wheel terminino con -any.whl o -linux_x86_64.whl e riprovare.

Verificare che sia possibile decomprimere il pacchetto di test senza errori. In questo esempio, il nome del pacchetto è test_bundle.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip test_bundle.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

   Se il pacchetto Appium Python è valido, troverai almeno un file *.whl* dipendente come i file evidenziati all'interno della directory. *wheelhouse* Il nome del file può essere diverso, ma dovrebbe terminare con *-any.whl* o*-linux_x86_64.whl*, che specifica la piattaforma. Qualsiasi altra piattaforma, come `windows` non è supportata.

```
.
|-- 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
```

   Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare la directory tests nel pacchetto di test. Decomprimere il pacchetto di test, verificare che la directory tests sia nel pacchetto e riprovare.

Verificare che sia possibile decomprimere il pacchetto di test senza errori. In questo esempio, il nome del pacchetto è test_bundle.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip test_bundle.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

   Se il pacchetto Appium Python è valido, troverai la *tests* directory all'interno della directory di lavoro.

```
.
|-- 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
```

   Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare un file di test valido nella struttura delle directory tests. Decomprimere il pacchetto di test, quindi aprire la directory di test, verificare che almeno un nome di file inizi o termini con la parola chiave "test" e riprovare.

Verificare che sia possibile decomprimere il pacchetto di test senza errori. In questo esempio, il nome del pacchetto è test_bundle.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip test_bundle.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Se il pacchetto Appium Python è valido, troverai la *tests* directory all'interno della directory di lavoro. Il nome del file può essere diverso, ma dovrebbe iniziare con *test_* o terminare con. *_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
```

Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare il file requirements.txt nel pacchetto di test. Decomprimere il pacchetto di test, verificare che il file requirements.txt sia nel pacchetto e riprovare.

Verificare che sia possibile decomprimere il pacchetto di test senza errori. In questo esempio, il nome del pacchetto è test_bundle.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip test_bundle.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

   Se il pacchetto Appium Python è valido, troverai il *requirements.txt* file all'interno della directory di lavoro.

```
.
|-- 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
```

   Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Trovata una versione di pytest inferiore alla versione minima 2.8.0 supportata. Modificare la versione pytest nel file requirements.txt e riprovare.

Verificare che sia possibile decomprimere il pacchetto di test senza errori. In questo esempio, il nome del pacchetto è test_bundle.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip test_bundle.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

   Dovresti trovare il *requirements.txt* file all'interno della directory di lavoro.

```
.
|-- 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. Per ottenere la versione pytest, eseguire il comando seguente:

```
$ grep "pytest" requirements.txt
```

   L'output dovrebbe essere come segue:

```
pytest==2.9.0
```

   Mostra la versione pytest, che in questo esempio è 2.9.0. Se il pacchetto Appium Python è valido, la versione di pytest deve essere maggiore o uguale a 2.8.0.

   Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS_FAIL

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile installare le wheel di dipendenza. Decomprimere il pacchetto di test, quindi aprire il file requirements.txt e la directory wheelhouse, verificare che le wheel di dipendenza specificate nel file requirements.txt corrispondano esattamente alle wheel di dipendenza all'interno della directory wheelhouse e riprovare.

Si consiglia vivamente di configurare un virtualenv Python per la creazione dei pacchetti di test. Ecco un esempio di flusso di creazione e attivazione di un ambiente virtuale utilizzando virtualenv Python:

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

Verificare che sia possibile decomprimere il pacchetto di test senza errori. In questo esempio, il nome del pacchetto è test_bundle.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip test_bundle.zip
   ```

2. Per eseguire test sull'installazione dei file wheel, eseguire il comando seguente:

   ```
   $ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
   requirements.txt
   ```

   Un pacchetto Appium Python valido dovrebbe produrre un output come il seguente:

   ```
   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. Per disattivare l'ambiente virtuale, eseguire il comando seguente:

```
$ deactivate
```

Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile raccogliere i test nella directory tests. Decomprimere il pacchetto di test, verificare che il pacchetto di test sia valido eseguendo il comando `py.test --collect-only <path to your tests directory>` e riprovare quando il comando non stampa più errori.

Si consiglia vivamente di configurare un virtualenv Python per la creazione dei pacchetti di test. Ecco un esempio di flusso di creazione e attivazione di un ambiente virtuale utilizzando virtualenv Python:

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

Verificare che sia possibile decomprimere il pacchetto di test senza errori. In questo esempio, il nome del pacchetto è test_bundle.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip test_bundle.zip
```

2. Per installare i file wheel, eseguire il comando seguente:

```
$ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
requirements.txt
```

3. Per raccogliere i test, eseguire il comando seguente:

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

Un pacchetto Appium Python valido dovrebbe produrre un output come il seguente:

```
==================== 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. Per disattivare l'ambiente virtuale, eseguire il comando seguente:

```
$ deactivate
```

Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEELS_INSUFFICIENT

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Non siamo riusciti a trovare abbastanza dipendenze tra le ruote nella directory wheelhouse. Decomprimi il pacchetto di test, quindi apri la cartella Wheelhouse. Verificate di avere tutte le dipendenze delle ruote specificate nel file requirements.txt.

Verificare che sia possibile decomprimere il pacchetto di test senza errori. In questo esempio, il nome del pacchetto è test_bundle.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip test_bundle.zip
```

2. Controlla la lunghezza del *requirements.txt* file e il numero di file *.whl* dipendenti nella directory wheelhouse:

```
$ cat requirements.txt | egrep "." |wc -l
    12
$ ls wheelhouse/ | egrep ".+\.whl" | wc -l
    11
```

   Se il numero di file *.whl* dipendenti è inferiore al numero di righe non vuote del *requirements.txt* file, devi assicurarti di quanto segue:

   - Esiste un file *.whl* dipendente corrispondente a ciascuna riga del *requirements.txt* file.
   - Non ci sono altre righe nel *requirements.txt* file che contengono informazioni diverse dai nomi dei pacchetti di dipendenza.
   - Nessun nome di dipendenza viene duplicato su più righe del *requirements.txt* file, in modo che due righe del file possano corrispondere a un *.whl* file dipendente.

   AWS Device Farm non supporta righe nel *requirements.txt* file che non corrispondono direttamente ai pacchetti di dipendenza, come le righe che specificano le opzioni globali per il `pip install` comando. Consulta il formato del file dei requisiti per un elenco di opzioni globali.

   Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

Risoluzione dei problemi dei test delle applicazioni web Appium Python in AWS Device Farm

Il seguente argomento elenca i messaggi di errore che si verificano durante il caricamento dei test per applicazioni Web di Appium Python e consiglia soluzioni alternative per risolvere ogni errore.

APPIUM_WEB_PYTHON_TEST_PACKAGE_UNZIP_FAILED

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile aprire il file ZIP del test Appium. Verificare che il file sia valido e riprovare.

Verificare che sia possibile decomprimere il pacchetto di test senza errori. In questo esempio, il nome del pacchetto è test_bundle.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip test_bundle.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Un pacchetto Appium Python valido dovrebbe produrre un output come il seguente:

   ```
   .
   |-- 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
   ```

   Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare un file wheel di dipendenza nella struttura delle directory wheelhouse. Decomprimere il pacchetto di test, quindi aprire la directory wheelhouse, verificare che nella directory si trovi almeno un file wheel e riprovare.

Verificare che sia possibile decomprimere il pacchetto di test senza errori. In questo esempio, il nome del pacchetto è test_bundle.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip test_bundle.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il pacchetto Appium Python è valido, troverai almeno un file *.whl* dipendente come i file evidenziati all'interno della directory. *wheelhouse*

   ```
   .
   |-- 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
   ```

   Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PLATFORM

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Trovato almeno un file wheel che specificava una piattaforma non supportata. Decomprimere il pacchetto di test, quindi aprire la directory wheelhouse, verificare che i nomi dei file wheel terminino con -any.whl o -linux_x86_64.whl e riprovare.

Verificare che sia possibile decomprimere il pacchetto di test senza errori. In questo esempio, il nome del pacchetto è test_bundle.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip test_bundle.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il pacchetto Appium Python è valido, troverai almeno un file *.whl* dipendente come i file evidenziati all'interno della directory. *wheelhouse* Il nome del file può essere diverso, ma dovrebbe terminare con *-any.whl* o*-linux_x86_64.whl*, che specifica la piattaforma. Qualsiasi altra piattaforma, come `windows` non è supportata.

   ```
   .
   |-- 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
   ```

   Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare la directory tests nel pacchetto di test. Decomprimere il pacchetto di test, verificare che la directory tests sia nel pacchetto e riprovare.

Verificare che sia possibile decomprimere il pacchetto di test senza errori. In questo esempio, il nome del pacchetto è test_bundle.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip test_bundle.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il pacchetto Appium Python è valido, troverai la *`tests`* directory all'interno della directory di lavoro.

   ```
   .
   |-- 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
   ```

   Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare un file di test valido nella struttura delle directory tests. Decomprimere il pacchetto di test, quindi aprire la directory di test, verificare che almeno un nome di file inizi o termini con la parola chiave "test" e riprovare.

Verificare che sia possibile decomprimere il pacchetto di test senza errori. In questo esempio, il nome del pacchetto è test_bundle.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip test_bundle.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il pacchetto Appium Python è valido, troverai la *tests* directory all'interno della directory di lavoro. Il nome del file può essere diverso, ma dovrebbe iniziare con *test_* o terminare con. *_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
   ```

   Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISS

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare il file requirements.txt nel pacchetto di test. Decomprimere il pacchetto di test, verificare che il file requirements.txt sia nel pacchetto e riprovare.

Verificare che sia possibile decomprimere il pacchetto di test senza errori. In questo esempio, il nome del pacchetto è test_bundle.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip test_bundle.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il pacchetto Appium Python è valido, troverai il *`requirements.txt`* file all'interno della directory di lavoro.

   ```
   .
   |-- 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
   ```

   Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Trovata una versione di pytest inferiore alla versione minima 2.8.0 supportata. Modificare la versione pytest nel file requirements.txt e riprovare.

Verificare che sia possibile decomprimere il pacchetto di test senza errori. In questo esempio, il nome del pacchetto è test_bundle.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip test_bundle.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Dovresti trovare il *requirement.txt* file all'interno della directory di lavoro.

   ```
   .
   |-- 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. Per ottenere la versione pytest, eseguire il comando seguente:

   ```
   $ grep "pytest" requirements.txt
   ```

L'output dovrebbe essere come segue:

```
pytest==2.9.0
```

Mostra la versione pytest, che in questo esempio è 2.9.0. Se il pacchetto Appium Python è valido, la versione di pytest deve essere maggiore o uguale a 2.8.0.

Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile installare le wheel di dipendenza. Decomprimere il pacchetto di test, quindi aprire il file requirements.txt e la directory wheelhouse, verificare che le wheel di dipendenza specificate nel file requirements.txt corrispondano esattamente alle wheel di dipendenza all'interno della directory wheelhouse e riprovare.

Si consiglia vivamente di configurare un virtualenv Python per la creazione dei pacchetti di test. Ecco un esempio di flusso di creazione e attivazione di un ambiente virtuale utilizzando virtualenv Python:

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

Verificare che sia possibile decomprimere il pacchetto di test senza errori. In questo esempio, il nome del pacchetto è test_bundle.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip test_bundle.zip
   ```

2. Per eseguire test sull'installazione dei file wheel, eseguire il comando seguente:

```
$ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
requirements.txt
```

Un pacchetto Appium Python valido dovrebbe produrre un output come il seguente:

```
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. Per disattivare l'ambiente virtuale, eseguire il comando seguente:

```
$ deactivate
```

Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile raccogliere i test nella directory tests. Decomprimere il pacchetto di test, verificare che il pacchetto di test sia valido eseguendo il comando "py.test --collect-only <path to your tests directory>" e riprovare quando il comando non stampa più errori.

Si consiglia vivamente di configurare un virtualenv Python per la creazione dei pacchetti di test. Ecco un esempio di flusso di creazione e attivazione di un ambiente virtuale utilizzando virtualenv Python:

```
$ virtualenv workspace
```

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

Verificare che sia possibile decomprimere il pacchetto di test senza errori. In questo esempio, il nome del pacchetto è test_bundle.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip test_bundle.zip
   ```

2. Per installare i file wheel, eseguire il comando seguente:

   ```
   $ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
   requirements.txt
   ```

3. Per raccogliere i test, eseguire il comando seguente:

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

   Un pacchetto Appium Python valido dovrebbe produrre un output come il seguente:

   ```
   ==================== 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. Per disattivare l'ambiente virtuale, eseguire il comando seguente:

   ```
   $ deactivate
   ```

   Per ulteriori informazioni, consulta Esegui automaticamente i test Appium in Device Farm.

Risoluzione dei problemi relativi ai test di strumentazione in AWS Device Farm

Il seguente argomento elenca i messaggi di errore che si verificano durante il caricamento dei test di Instrumentation e consiglia soluzioni alternative per risolvere ogni errore.

> **Note**
>
> Per considerazioni importanti sull'utilizzo dei test di strumentazione in AWS Device Farm, consulta. Strumentazione per Android e AWS Device Farm

INSTRUMENTATION_TEST_PACKAGE_UNZIP_FAILED

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

```
Warning: We could not open your test APK file. Please verify that the file is valid and
    try again.
```

Verificare che sia possibile decomprimere il pacchetto di test senza errori. Nell'esempio seguente, il nome del pacchetto è Test-unaligned.apk. app-debug-android

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip app-debug-androidTest-unaligned.apk
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Un pacchetto di test Instrumentation valido produce un output come il seguente:

   ```
   .
   |-- AndroidManifest.xml
   |--  classes.dex
   |-- resources.arsc
   |-- LICENSE-junit.txt
   |-- junit (directory)
   ```

```
`-- META-INF (directory)
```

Per ulteriori informazioni, consulta Strumentazione per Android e AWS Device Farm.

INSTRUMENTATION_TEST_PACKAGE_AAPT_DEBUG_BADGING_FAILED

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

```
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.
```

Durante il processo di convalida del caricamento, Device Farm analizza le informazioni dall'output del `aapt debug badging <path to your package>` comando.

Controllare che questo comando possa essere eseguito correttamente sul pacchetto di test Instrumentation

Nell'esempio seguente, il nome del pacchetto è app-debug-android Test-unaligned.apk.

- Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

  ```
  $ aapt debug badging app-debug-androidTest-unaligned.apk
  ```

  Un pacchetto di test Instrumentation valido produce un output come il seguente:

  ```
  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'
```

Per ulteriori informazioni, consulta Strumentazione per Android e AWS Device Farm.

INSTRUMENTATION_TEST_PACKAGE_INSTRUMENTATION_RUNNER_VALU

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

```
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."
```

Durante il processo di convalida del caricamento, Device Farm analizza il valore dell'instrumentation runner dall'albero di analisi XML per un file XML contenuto nel pacchetto. Utilizzare il seguente comando: `aapt dump xmltree <path to your package> AndroidManifest.xml`.

Controllare che sia possibile eseguire questo comando nel pacchetto di test Instrumentation e sia possibile trovare il valore instrumentation.

Nell'esempio seguente, il nome del pacchetto è Test-unaligned.apk. app-debug-android

- Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

  ```
  $ aapt dump xmltree app-debug-androidTest-unaligned.apk AndroidManifest.xml | grep
   -A5 "instrumentation"
  ```

  Un pacchetto di test Instrumentation valido produce un output come il seguente:

  ```
  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
```

Per ulteriori informazioni, consulta Strumentazione per Android e AWS Device Farm.

INSTRUMENTATION_TEST_PACKAGE_AAPT_DUMP_XMLTREE_FAILED

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

```
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.
```

Durante il processo di convalida del caricamento, Device Farm analizza le informazioni dall'albero di analisi XML per un file XML contenuto nel pacchetto utilizzando il seguente comando:. `aapt dump xmltree <path to your package> AndroidManifest.xml`

Controllare che questo comando possa essere eseguito correttamente sul pacchetto di test Instrumentation.

Nel seguente esempio, il nome del pacchetto è Test-unaligned.apk. app-debug-android

- Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

  ```
  $ aapt dump xmltree app-debug-androidTest-unaligned.apk AndroidManifest.xml
  ```

  Un pacchetto di test Instrumentation valido produce un output come il seguente:

  ```
  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")
```

Per ulteriori informazioni, consulta Strumentazione per Android e AWS Device Farm.

INSTRUMENTATION_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISS

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

```
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."
```

Durante il processo di convalida del caricamento, Device Farm analizza il valore del nome del pacchetto dall'output del seguente comando:. `aapt debug badging <path to your package>`

Controllare che sia possibile eseguire questo comando nel pacchetto di test Instrumentation e che sia possibile trovare il valore del nome del pacchetto.

Nell'esempio seguente, il nome del pacchetto è app-debug-android Test-unaligned.apk.

- Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ aapt debug badging app-debug-androidTest-unaligned.apk | grep "package: name="
```

Un pacchetto di test Instrumentation valido produce un output come il seguente:

```
package: name='com.amazon.aws.adf.android.referenceapp.test' versionCode=''
 versionName='' platformBuildVersionName='5.1.1-1819727'
```

Per ulteriori informazioni, consulta Strumentazione per Android e AWS Device Farm.

Risoluzione dei problemi relativi ai test delle applicazioni iOS in AWS Device Farm

Il seguente argomento elenca i messaggi di errore che si verificano durante il caricamento dei test di applicazioni iOS e consiglia soluzioni alternative per risolvere ogni errore.

> **Note**
>
> Le seguenti istruzioni si basano su Linux x86_64 e Mac.

IOS_APP_UNZIP_FAILED

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile aprire l'applicazione. Verificare che il file sia valido e riprovare.

Verificare che sia possibile decomprimere il pacchetto dell'applicazione senza errori. Nell'esempio seguente, il nome del pacchetto è AWSDeviceFarmi OSReference App.ipa.

1. Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Un pacchetto di un'applicazione iOS valido dovrebbe produrre un output come il seguente:

```
.
`-- Payload (directory)
        `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                    |-- Info.plist
                    `-- (any other files)
```

Per ulteriori informazioni, consulta Test iOS in AWS Device Farm.

IOS_APP_PAYLOAD_DIR_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare la directory Payload nell'applicazione. Decomprimere l'applicazione, verificare che la directory Payload sia nel pacchetto e riprovare.

Nell'esempio seguente, il nome del pacchetto è Farmi App.IPA. AWSDevice OSReference

1. Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Se il pacchetto dell'applicazione iOS è valido, troverai la *Payload* directory all'interno della directory di lavoro.

```
.
`-- Payload (directory)
        `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                        |-- Info.plist
                        `-- (any other files)
```

Per ulteriori informazioni, consulta Test iOS in AWS Device Farm.

IOS_APP_APP_DIR_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare la directory .app all'interno della directory Payload. Decomprimere l'applicazione, quindi aprire la directory Payload, verificare che la directory .app si trovi all'interno della directory e riprovare.

Nell'esempio seguente, il nome del pacchetto è AWSDeviceFarmi OSReference App.IPA.

1. Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il pacchetto dell'applicazione iOS è valido, all'interno della *.app* directory troverai una directory come *AWSDeviceFarmiOSReferenceApp.app* nel nostro esempio. *Payload*

   ```
   .
   ```

```
`-- Payload (directory)
        `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                        |-- Info.plist
                        `-- (any other files)
```

Per ulteriori informazioni, consulta Test iOS in AWS Device Farm.

IOS_APP_PLIST_FILE_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare il file Info.plist all'interno della directory .app. Decomprimere l'applicazione, quindi aprire la directory .app, verificare che il file Info.plist si trovi all'interno della directory e riprovare.

Nell'esempio seguente, il nome del pacchetto è AWSDeviceFarmi OSReference App.ipa.

1. Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il pacchetto dell'applicazione iOS è valido, troverai il *Info.plist* file all'interno della *.app* directory come *AWSDeviceFarmiOSReferenceApp.app* nel nostro esempio.

   ```
   .
   `-- Payload (directory)
           `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                           |-- Info.plist
                           `-- (any other files)
   ```

Per ulteriori informazioni, consulta Test iOS in AWS Device Farm.

IOS_APP_CPU_ARCHITECTURE_VALUE_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare il valore dell'architettura CPU nel file Info.plist. Decomprimi l'applicazione, quindi apri il file Info.plist nella directory.app, verifica che sia specificata la chiave "UIRequiredDeviceCapabilities" e riprova.

Nell'esempio seguente, il nome del pacchetto è Farmi App.IPA. AWSDevice OSReference

1. Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *AWSDeviceFarmiOSReferenceApp.app* nel nostro esempio:

   ```
   .
   `-- Payload (directory)
           `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                       |-- Info.plist
                       `-- (any other files)
   ```

3. Per individuare il valore dell'architettura CPU, aprire Info.plist utilizzando Xcode o Python.

   Per Python, installare il modulo biplist eseguendo il seguente comando:

```
$ pip install biplist
```

4. Aprire quindi Python ed eseguire il seguente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['UIRequiredDeviceCapabilities']
```

Un pacchetto di un'applicazione iOS valido dovrebbe produrre un output come il seguente:

```
['armv7']
```

Per ulteriori informazioni, consulta Test iOS in AWS Device Farm.

IOS_APP_PLATFORM_VALUE_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare il valore della piattaforma nel file Info.plist. Decomprimi l'applicazione e quindi apri il file Info.plist nella directory.app, verifica che sia specificata la chiave "CFBundleSupportedPlatforms" e riprova.

Nell'esempio seguente, il nome del pacchetto è Farmi App.IPA. AWSDevice OSReference

1. Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *AWSDeviceFarmiOSReferenceApp.app* nel nostro esempio:

```
.
`-- Payload (directory)
        `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                        |-- Info.plist
                        `-- (any other files)
```

3. Per individuare il valore della piattaforma, aprire Info.plist utilizzando Xcode o Python.

   Per Python, installare il modulo biplist eseguendo il seguente comando:

```
$ pip install biplist
```

4. Aprire quindi Python ed eseguire il seguente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['CFBundleSupportedPlatforms']
```

   Un pacchetto di un'applicazione iOS valido dovrebbe produrre un output come il seguente:

```
['iPhoneOS']
```

   Per ulteriori informazioni, consulta Test iOS in AWS Device Farm.

IOS_APP_WRONG_PLATFORM_DEVICE_VALUE

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Trovato un valore per il dispositivo della piattaforma errato nel file Info.plist. Decomprimi l'applicazione e poi apri il file Info.plist nella directory.app, verifica che il valore della chiave "CFBundleSupportedPlatforms" non contenga la parola chiave «simulator» e riprova.

Nell'esempio seguente, il nome del pacchetto è Farmi App.IPA. AWSDevice OSReference

1. Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *AWSDeviceFarmiOSReferenceApp.app* nel nostro esempio:

```
.
`-- Payload (directory)
        `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                        |-- Info.plist
                        `-- (any other files)
```

3. Per individuare il valore della piattaforma, aprire Info.plist utilizzando Xcode o Python.

Per Python, installare il modulo biplist eseguendo il seguente comando:

```
$ pip install biplist
```

4. Aprire quindi Python ed eseguire il seguente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['CFBundleSupportedPlatforms']
```

Un pacchetto di un'applicazione iOS valido dovrebbe produrre un output come il seguente:

```
['iPhoneOS']
```

Se l'applicazione iOS è valida, il valore non deve contenere la parola chiave: `simulator`.

Per ulteriori informazioni, consulta Test iOS in AWS Device Farm.

IOS_APP_FORM_FACTOR_VALUE_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare il valore del fattore di forma nel file Info.plist. Decomprimi l'applicazione e quindi apri il file Info.plist nella directory.app, verifica che sia specificata la chiave "UIDeviceFamiglia» e riprova.

Nell'esempio seguente, il nome del pacchetto è Farmi App.IPA. AWSDevice OSReference

1. Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *AWSDeviceFarmiOSReferenceApp.app* nel nostro esempio:

   ```
   .
   `-- Payload (directory)
           `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                       |-- Info.plist
                       `-- (any other files)
   ```

3. Per individuare il valore del fattore di forma, aprire Info.plist utilizzando Xcode o Python.

   Per Python, installare il modulo biplist eseguendo il seguente comando:

```
$ pip install biplist
```

4. Aprire quindi Python ed eseguire il seguente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['UIDeviceFamily']
```

Un pacchetto di un'applicazione iOS valido dovrebbe produrre un output come il seguente:

```
[1, 2]
```

Per ulteriori informazioni, consulta Test iOS in AWS Device Farm.

IOS_APP_PACKAGE_NAME_VALUE_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare il valore del nome del pacchetto nel file Info.plist. Decomprimi l'applicazione e quindi apri il file Info.plist nella directory.app, verifica che sia specificata la chiave "CFBundleIdentifier» e riprova.

Nell'esempio seguente, il nome del pacchetto è Farmi App.IPA. AWSDevice OSReference

1. Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *AWSDeviceFarmiOSReferenceApp.app* nel nostro esempio:

```
.
`-- Payload (directory)
        `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                        |-- Info.plist
                        `-- (any other files)
```

3. Per individuare il valore del nome del pacchetto, aprire Info.plist utilizzando Xcode o Python.

   Per Python, installare il modulo biplist eseguendo il seguente comando:

   ```
   $ pip install biplist
   ```

4. Aprire quindi Python ed eseguire il seguente comando:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
   Info.plist')
   print info_plist['CFBundleIdentifier']
   ```

   Un pacchetto di un'applicazione iOS valido dovrebbe produrre un output come il seguente:

   ```
   Amazon.AWSDeviceFarmiOSReferenceApp
   ```

   Per ulteriori informazioni, consulta Test iOS in AWS Device Farm.

IOS_APP_EXECUTABLE_VALUE_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare il valore dell'eseguibile nel file Info.plist. Decomprimi l'applicazione e quindi apri il file Info.plist nella directory.app, verifica che sia specificata la chiave "CFBundleExecutable» e riprova.

Nell'esempio seguente, il nome del pacchetto è Farmi App.IPA. AWSDevice OSReference

1. Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *AWSDeviceFarmiOSReferenceApp.app* nel nostro esempio:

```
.
`-- Payload (directory)
        `-- AWSDeviceFarmiOSReferenceApp.app (directory)
                        |-- Info.plist
                        `-- (any other files)
```

3. Per individuare il valore dell'eseguibile, aprire Info.plist utilizzando Xcode o Python.

Per Python, installare il modulo biplist eseguendo il seguente comando:

```
$ pip install biplist
```

4. Aprire quindi Python ed eseguire il seguente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['CFBundleExecutable']
```

Un pacchetto di un'applicazione iOS valido dovrebbe produrre un output come il seguente:

```
AWSDeviceFarmiOSReferenceApp
```

Per ulteriori informazioni, consulta Test iOS in AWS Device Farm.

XCTest Test di risoluzione dei problemi in AWS Device Farm

L'argomento seguente elenca i messaggi di errore che si verificano durante il caricamento dei XCTest test e consiglia soluzioni alternative per risolvere ogni errore.

> **Note**
>
> Le istruzioni seguenti presumono l'utilizzo di un sistema MacOS.

XCTEST_TEST_PACKAGE_UNZIP_FAILED

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile aprire il file ZIP del test. Verificare che il file sia valido e riprovare.

Verificare che sia possibile decomprimere il pacchetto dell'applicazione senza errori. Nell'esempio seguente, il nome del pacchetto è .xctest-1.zip. swiftExampleTests

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip swiftExampleTests.xctest-1.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Un XCTest pacchetto valido dovrebbe produrre un output simile al seguente:

   ```
   .
   `-- swiftExampleTests.xctest (directory)
                |-- Info.plist
                `-- (any other files)
   ```

   Per ulteriori informazioni, consulta Integrazione di Device Farm con XCTest iOS.

XCTEST_TEST_PACKAGE_XCTEST_DIR_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare la directory .xctest nel pacchetto di test. Decomprimere il pacchetto di test, verificare che la directory .xctest sia nel pacchetto e riprovare.

Nell'esempio seguente, il nome del pacchetto è swiftExampleTests.xctest-1.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip swiftExampleTests.xctest-1.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il XCTest pacchetto è valido, troverai una directory con un nome simile a quello contenuto nella directory di *swiftExampleTests.xctest* lavoro. Il nome dovrebbe terminare con*.xctest*.

   ```
   .
   `-- swiftExampleTests.xctest (directory)
                |-- Info.plist
                `-- (any other files)
   ```

   Per ulteriori informazioni, consulta Integrazione di Device Farm con XCTest iOS.

XCTEST_TEST_PACKAGE_PLIST_FILE_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare il file Info.plist all'interno della directory .xctest. Decomprimere il pacchetto di test, quindi aprire la directory .xctest, verificare che il file Info.plist si trovi nella directory e riprovare.

Nell'esempio seguente, il nome del pacchetto è swiftExampleTests.xctest-1.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip swiftExampleTests.xctest-1.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il XCTest pacchetto è valido, il file si trova all'interno della *Info.plist* directory. *.xctest* Nel nostro esempio seguente, viene chiamata la directory*swiftExampleTests.xctest*.

   ```
   .
   `-- swiftExampleTests.xctest (directory)
                |-- Info.plist
                `-- (any other files)
   ```

   Per ulteriori informazioni, consulta Integrazione di Device Farm con XCTest iOS.

XCTEST_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare il valore del nome del pacchetto nel file Info.plist. Decomprimi il pacchetto di test, quindi apri il file Info.plist, verifica che sia specificata la chiave "CFBundleIdentifier» e riprova.

Nell'esempio seguente, il nome del pacchetto è .xctest-1.zip. swiftExampleTests

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

   Dovresti trovare il *Info.plist* file all'interno di una *.xctest* directory come nel nostro esempio: *swiftExampleTests.xctest*

```
.
`-- swiftExampleTests.xctest (directory)
              |-- Info.plist
              `-- (any other files)
```

3. Per individuare il valore del nome del pacchetto, aprire Info.plist utilizzando Xcode o Python.

   Per Python, installare il modulo biplist eseguendo il seguente comando:

```
$ pip install biplist
```

4. Aprire quindi Python ed eseguire il seguente comando:

```
import biplist
info_plist = biplist.readPlist('swiftExampleTests.xctest/Info.plist')
print info_plist['CFBundleIdentifier']
```

   Un pacchetto XCtest applicativo valido dovrebbe produrre un output simile al seguente:

```
com.amazon.kanapka.swiftExampleTests
```

   Per ulteriori informazioni, consulta Integrazione di Device Farm con XCTest iOS.

XCTEST_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

> **Warning**
>
> Impossibile trovare il valore dell'eseguibile nel file Info.plist. Decomprimi il pacchetto di test, quindi apri il file Info.plist, verifica che sia specificata la chiave "CFBundleEseguibile» e riprova.

Nell'esempio seguente, il nome del pacchetto è .xctest-1.zip. swiftExampleTests

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip swiftExampleTests.xctest-1.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Dovresti trovare il *Info.plist* file all'interno di una *.xctest* directory come nel nostro esempio: *swiftExampleTests.xctest*

   ```
   .
   `-- swiftExampleTests.xctest (directory)
                |-- Info.plist
                `-- (any other files)
   ```

3. Per individuare il valore del nome del pacchetto, aprire Info.plist utilizzando Xcode o Python.

   Per Python, installare il modulo biplist eseguendo il seguente comando:

   ```
   $ pip install biplist
   ```

4. Aprire quindi Python ed eseguire il seguente comando:

   ```
   import biplist
   info_plist = biplist.readPlist('swiftExampleTests.xctest/Info.plist')
   ```

```
print info_plist['CFBundleExecutable']
```

Un pacchetto XCtest applicativo valido dovrebbe produrre un output simile al seguente:

```
swiftExampleTests
```

Per ulteriori informazioni, consulta Integrazione di Device Farm con XCTest iOS.

Risoluzione dei problemi dei test XCTest dell'interfaccia utente in AWS Device Farm

Il seguente argomento elenca i messaggi di errore che si verificano durante il caricamento dei test dell' XCTest interfaccia utente e consiglia soluzioni alternative per risolvere ogni errore.

> **Note**
>
> Le seguenti istruzioni si basano su Linux x86_64 e Mac.

XCTEST_UI_TEST_PACKAGE_UNZIP_FAILED

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

```
We could not open your test IPA file. Please verify that the file is valid and try again.
```

Verificare che sia possibile decomprimere il pacchetto dell'applicazione senza errori. Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

Un pacchetto di un'applicazione iOS valido dovrebbe produrre un output come il seguente:

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                        |-- Info.plist
                        |-- Plugins (directory)
                        |       `swift-sampleUITests.xctest (directory)
                        |                         |-- Info.plist
                        |                         `-- (any other files)
                        `-- (any other files)
```

Per ulteriori informazioni, consulta Integrazione dell' XCTest interfaccia utente per iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

`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.`

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il pacchetto XCTest UI è valido, troverai la *Payload* directory all'interno della directory di lavoro.

   ```
   .
   `-- Payload (directory)
   ```

```
        `-- swift-sampleUITests-Runner.app (directory)
                    |-- Info.plist
                    |-- Plugins (directory)
                    |       `swift-sampleUITests.xctest (directory)
                    |                       |-- Info.plist
                    |                       `-- (any other files)
                    `-- (any other files)
```

Per ulteriori informazioni, consulta Integrazione dell' XCTest interfaccia utente per iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_APP_DIR_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

`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.`

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il pacchetto XCTest UI è valido, all'interno della *.app* directory troverai una directory come *swift-sampleUITests-Runner.app* nel nostro esempio. *Payload*

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                       |-- Info.plist
   ```

```
                        |                       `-- (any other files)
                        `-- (any other files)
```

Per ulteriori informazioni, consulta Integrazione dell' XCTest interfaccia utente per iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_PLUGINS_DIR_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

```
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.
```

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il pacchetto XCTest UI è valido, troverai la *Plugins* directory all'interno di una *.app* directory. Nel nostro esempio, viene chiamata la directory*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)
   ```

Per ulteriori informazioni, consulta Integrazione dell' XCTest interfaccia utente per iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_XCTEST_DIR_MISSING_IN_PLUGINS_DIR

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

```
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.
```

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il pacchetto XCTest UI è valido, troverai una *.xctest* directory all'interno della *Plugins* directory. Nel nostro esempio, viene chiamata la directory*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)
   ```

   Per ulteriori informazioni, consulta Integrazione dell' XCTest interfaccia utente per iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

```
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.
```

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il pacchetto XCTest UI è valido, troverai il *Info.plist* file all'interno della *.app* directory. Nel nostro esempio seguente, viene chiamata la directory*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)
   ```

   Per ulteriori informazioni, consulta Integrazione dell' XCTest interfaccia utente per iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING_IN_XCTEST_DIR

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

```
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.
```

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il pacchetto XCTest UI è valido, troverai il *`Info.plist`* file all'interno della *`.xctest`* directory. Nel nostro esempio seguente, viene chiamata la directory*`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)
   ```

   Per ulteriori informazioni, consulta Integrazione dell' XCTest interfaccia utente per iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_CPU_ARCHITECTURE_VALUE_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

```
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.
```

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *swift-sampleUITests-Runner.app* nel nostro esempio:

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                       |-- Info.plist
                       |                       `-- (any other files)
                       `-- (any other files)
   ```

3. Per individuare il valore dell'architettura CPU, aprire Info.plist utilizzando Xcode o Python.

   Per Python, installare il modulo biplist eseguendo il seguente comando:

   ```
   $ pip install biplist
   ```

4. Aprire quindi Python ed eseguire il seguente comando:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/
   Info.plist')
   print info_plist['UIRequiredDeviceCapabilities']
   ```

   Un pacchetto XCtest UI valido dovrebbe produrre un output simile al seguente:

```
['armv7']
```

Per ulteriori informazioni, consulta Integrazione dell' XCTest interfaccia utente per iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_PLATFORM_VALUE_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

```
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.
```

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *swift-sampleUITests-Runner.app* nel nostro esempio:

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                         |-- Info.plist
                       |                         `-- (any other files)
                       `-- (any other files)
   ```

3. Per individuare il valore della piattaforma, aprire Info.plist utilizzando Xcode o Python.

Per Python, installare il modulo biplist eseguendo il seguente comando:

```
$ pip install biplist
```

4. Aprire quindi Python ed eseguire il seguente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['CFBundleSupportedPlatforms']
```

Un pacchetto XCtest UI valido dovrebbe produrre un output simile al seguente:

```
['iPhoneOS']
```

Per ulteriori informazioni, consulta Integrazione dell' XCTest interfaccia utente per iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_WRONG_PLATFORM_DEVICE_VALUE

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

`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.`

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swift-sample-UI.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *swift-sampleUITests-Runner.app* nel nostro esempio:

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                        |-- Info.plist
                        |-- Plugins (directory)
                        |       `swift-sampleUITests.xctest (directory)
                        |                         |-- Info.plist
                        |                         `-- (any other files)
                        `-- (any other files)
```

3. Per individuare il valore della piattaforma, aprire Info.plist utilizzando Xcode o Python.

   Per Python, installare il modulo biplist eseguendo il seguente comando:

   ```
   $ pip install biplist
   ```

4. Aprire quindi Python ed eseguire il seguente comando:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
   print info_plist['CFBundleSupportedPlatforms']
   ```

   Un pacchetto XCtest UI valido dovrebbe produrre un output simile al seguente:

   ```
   ['iPhoneOS']
   ```

   Se il pacchetto XCTest UI è valido, il valore non deve contenere la parola chiave`simulator`.

   Per ulteriori informazioni, consulta Integrazione dell' XCTest interfaccia utente per iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_FORM_FACTOR_VALUE_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

```
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.
```

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *swift-sampleUITests-Runner.app* nel nostro esempio:

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                         |-- Info.plist
                       |                         `-- (any other files)
                       `-- (any other files)
   ```

3. Per individuare il valore del fattore di forma, aprire Info.plist utilizzando Xcode o Python.

   Per Python, installare il modulo biplist eseguendo il seguente comando:

   ```
   $ pip install biplist
   ```

4. Aprire quindi Python ed eseguire il seguente comando:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
   print info_plist['UIDeviceFamily']
   ```

   Un pacchetto XCtest UI valido dovrebbe produrre un output simile al seguente:

```
[1, 2]
```

Per ulteriori informazioni, consulta Integrazione dell' XCTest interfaccia utente per iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

`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.`

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *swift-sampleUITests-Runner.app* nel nostro esempio:

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                         |-- Info.plist
                       |                         `-- (any other files)
                       `-- (any other files)
   ```

3. Per individuare il valore del nome del pacchetto, aprire Info.plist utilizzando Xcode o Python.

Per Python, installare il modulo biplist eseguendo il seguente comando:

```
$ pip install biplist
```

4. Aprire quindi Python ed eseguire il seguente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['CFBundleIdentifier']
```

Un pacchetto XCtest UI valido dovrebbe produrre un output simile al seguente:

```
com.apple.test.swift-sampleUITests-Runner
```

Per ulteriori informazioni, consulta Integrazione dell' XCTest interfaccia utente per iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

```
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.
```

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swift-sample-UI.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *swift-sampleUITests-Runner.app* nel nostro esempio:

```
.
`-- Payload (directory)
        `-- swift-sampleUITests-Runner.app (directory)
                        |-- Info.plist
                        |-- Plugins (directory)
                        |       `swift-sampleUITests.xctest (directory)
                        |                         |-- Info.plist
                        |                         `-- (any other files)
                        `-- (any other files)
```

3. Per individuare il valore dell'eseguibile, aprire Info.plist utilizzando Xcode o Python.

   Per Python, installare il modulo biplist eseguendo il seguente comando:

   ```
   $ pip install biplist
   ```

4. Aprire quindi Python ed eseguire il seguente comando:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
   print info_plist['CFBundleExecutable']
   ```

   Un pacchetto XCtest UI valido dovrebbe produrre un output simile al seguente:

   ```
   XCTRunner
   ```

   Per ulteriori informazioni, consulta Integrazione dell' XCTest interfaccia utente per iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

```
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.
```

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *swift-sampleUITests-Runner.app* nel nostro esempio:

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                       |-- Info.plist
                       |                       `-- (any other files)
                       `-- (any other files)
   ```

3. Per individuare il valore del nome del pacchetto, aprire Info.plist utilizzando Xcode o Python.

   Per Python, installare il modulo biplist eseguendo il seguente comando:

   ```
   $ pip install biplist
   ```

4. Aprire quindi Python ed eseguire il seguente comando:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Plugins/
   swift-sampleUITests.xctest/Info.plist')
   print info_plist['CFBundleIdentifier']
   ```

   Un pacchetto XCtest UI valido dovrebbe produrre un output simile al seguente:

```
com.amazon.swift-sampleUITests
```

Per ulteriori informazioni, consulta Integrazione dell' XCTest interfaccia utente per iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_TEST_EXECUTABLE_VALUE_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

```
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.
```

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *swift-sampleUITests-Runner.app* nel nostro esempio:

   ```
   .
   `-- Payload (directory)
           `-- swift-sampleUITests-Runner.app (directory)
                       |-- Info.plist
                       |-- Plugins (directory)
                       |       `swift-sampleUITests.xctest (directory)
                       |                         |-- Info.plist
                       |                         `-- (any other files)
                       `-- (any other files)
   ```

3. Per individuare il valore dell'eseguibile, aprire Info.plist utilizzando Xcode o Python.

Per Python, installare il modulo biplist eseguendo il seguente comando:

```
$ pip install biplist
```

4. Aprire quindi Python ed eseguire il seguente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Plugins/
swift-sampleUITests.xctest/Info.plist')
print info_plist['CFBundleExecutable']
```

Un pacchetto XCtest UI valido dovrebbe produrre un output simile al seguente:

```
swift-sampleUITests
```

Per ulteriori informazioni, consulta Integrazione dell' XCTest interfaccia utente per iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_MULTIPLE_APP_DIRS

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

`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. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swift-sample-UI.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Se il pacchetto XCTest UI è valido, dovresti trovare solo una singola directory come nel nostro esempio all'interno del pacchetto di test.zip. `.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)
```

Per ulteriori informazioni, consulta Integrazione dell' XCTest interfaccia utente per iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_MULTIPLE_IPA_DIRS

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

`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. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swift-sample-UI.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Se il pacchetto XCTest UI è valido, dovresti trovare solo una singola directory come nel nostro esempio all'interno del pacchetto di test.zip. `.ipa` `sampleUITests.ipa`

```
.
`--swift-sample-UI.zip--(directory)
    `-- sampleUITests.ipa (directory)
            `-- Payload (directory)
                `-- swift-sampleUITests-Runner.app (directory)
```

```
    `-- (any other files)
```

Per ulteriori informazioni, consulta Integrazione dell' XCTest interfaccia utente per iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_BOTH_APP_AND_IPA_DIR_PRESENT

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

`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. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swift-sample-UI.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Se il pacchetto UI è valido, dovresti trovare una directory simile o una directory come nel nostro esempio all'interno del pacchetto di test.zip. XCTest `.ipa` `sampleUITests.ipa` `.app` `swift-sampleUITests-Runner.app` Puoi fare riferimento a un esempio di pacchetto di test XCTEST_UI valido nella nostra documentazione su. Integrazione dell' XCTest interfaccia utente per iOS con Device Farm

```
.
`--swift-sample-UI.zip--(directory)
    `-- sampleUITests.ipa (directory)
            `-- Payload (directory)
                `-- swift-sampleUITests-Runner.app (directory)
   `-- (any other files)
```

oppure

```
.
`--swift-sample-UI.zip--(directory)
```

```
    `-- swift-sampleUITests-Runner.app (directory)
            |-- Info.plist
            |-- Plugins (directory)
            `-- (any other files)
   `-- (any other files)
```

Per ulteriori informazioni, consulta Integrazione dell' XCTest interfaccia utente per iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_PRESENT_IN_ZIP

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

```
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. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

   ```
   $ unzip swift-sample-UI.zip
   ```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

   ```
   $ tree .
   ```

   Se il pacchetto UI è valido, non dovresti trovare una Payload Directory all'interno del pacchetto di test. 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)
   ```

Per ulteriori informazioni, consulta Integrazione dell' XCTest interfaccia utente per iOS con Device Farm.

Sicurezza in AWS Device Farm

La sicurezza del cloud AWS è la massima priorità. In qualità di AWS cliente, puoi beneficiare di un data center e di un'architettura di rete progettati per soddisfare i requisiti delle organizzazioni più sensibili alla sicurezza.

La sicurezza è una responsabilità condivisa tra AWS te e te. Il modello di responsabilità condivisa descrive questo aspetto come sicurezza del cloud e sicurezza nel cloud:

- Sicurezza del cloud: AWS è responsabile della protezione dell'infrastruttura che gestisce AWS i servizi nel AWS cloud. AWS ti fornisce anche servizi che puoi utilizzare in modo sicuro. I revisori esterni testano e verificano regolarmente l'efficacia della nostra sicurezza nell'ambito dei AWS Programmi di AWS conformità dei Programmi di conformità dei di . Per ulteriori informazioni sui programmi di conformità applicabili AWS Device Farm, consulta AWS Services in Scope by Compliance Program .
- Sicurezza nel cloud: la responsabilità dell'utente è determinata dal servizio AWS utilizzato. L'utente è anche responsabile di altri fattori, tra cui la riservatezza dei dati, i requisiti della propria azienda e le leggi e normative vigenti.

Questa documentazione aiuta a capire come applicare il modello di responsabilità condivisa quando si utilizza Device Farm. I seguenti argomenti mostrano come configurare Device Farm per soddisfare gli obiettivi di sicurezza e conformità. Imparerai anche a usare altri servizi AWS che ti aiutano a monitorare e proteggere le tue risorse Device Farm.

Argomenti

Gestione delle identità e degli accessi in AWS Device Farm

Destinatari

Il modo in cui utilizzi AWS Identity and Access Management (IAM) varia in base al tuo ruolo:

- Utente del servizio: richiedi le autorizzazioni all'amministratore se non riesci ad accedere alle funzionalità (consulta Risoluzione dei problemi di identità e accesso ad AWS Device Farm)
- Amministratore del servizio: determina l'accesso degli utenti e invia le richieste di autorizzazione (consulta Come funziona AWS Device Farm con IAM)
- Amministratore IAM: scrivi policy per gestire l'accesso (consulta Esempi di policy basate sull'identità di AWS Device Farm)

Autenticazione con identità

L'autenticazione è il modo in cui accedi AWS utilizzando le tue credenziali di identità. Devi autenticarti come utente IAM o assumendo un ruolo IAM. Utente root dell'account AWS

Puoi accedere come identità federata utilizzando credenziali provenienti da una fonte di identità come AWS IAM Identity Center (IAM Identity Center), autenticazione Single Sign-On o credenziali. Google/ Facebook Per ulteriori informazioni sull'accesso, consulta Come accedere all' Account AWS nella Guida per l'utente di Accedi ad AWS .

Per l'accesso programmatico, AWS fornisce un SDK e una CLI per firmare crittograficamente le richieste. Per ulteriori informazioni, consulta AWS Signature Version 4 per le richieste API nella Guida per l'utente di IAM.

Account AWS utente root

Quando si crea un Account AWS, si inizia con un'identità di accesso denominata utente Account AWS root che ha accesso completo a tutte Servizi AWS le risorse. Consigliamo vivamente di non utilizzare l'utente root per le attività quotidiane. Per le attività che richiedono le credenziali come utente root, consulta Attività che richiedono le credenziali dell'utente root nella Guida per l'utente di IAM.

Utenti IAM e gruppi

Un utente IAM è una identità che dispone di autorizzazioni specifiche per una singola persona o applicazione. Ti consigliamo di utilizzare credenziali temporanee invece di utenti IAM con credenziali

a lungo termine. Per ulteriori informazioni, consulta Richiedere agli utenti umani di utilizzare la federazione con un provider di identità per accedere AWS utilizzando credenziali temporanee nella Guida per l'utente IAM.

Un gruppo IAM specifica una raccolta di utenti IAM e semplifica la gestione delle autorizzazioni per gestire gruppi di utenti di grandi dimensioni. Per ulteriori informazioni, consulta Casi d'uso per utenti IAM nella Guida per l'utente di IAM.

Ruoli IAM

Un ruolo IAM è un'identità con autorizzazioni specifiche che fornisce credenziali temporanee. Puoi assumere un ruolo passando da un ruolo utente a un ruolo IAM (console) o chiamando un'operazione AWS CLI o AWS API. Per ulteriori informazioni, consulta Metodi per assumere un ruolo nella Guida per l'utente di IAM.

I ruoli IAM sono utili per l'accesso degli utenti federati, le autorizzazioni utente IAM temporanee, l'accesso multi-account, l'accesso multi-servizio e le applicazioni in esecuzione su Amazon EC2. Per maggiori informazioni, consultare Accesso a risorse multi-account in IAM nella Guida per l'utente IAM.

Come funziona AWS Device Farm con IAM

Prima di utilizzare IAM per gestire l'accesso a Device Farm, è necessario comprendere quali funzionalità IAM sono disponibili per l'uso con Device Farm. Per avere una visione di alto livello di come Device Farm e altri AWS servizi funzionano con IAM, consulta AWS Services That Work with IAM nella IAM User Guide.

Argomenti

- Politiche basate sull'identità di Device Farm
- Politiche basate sulle risorse di Device Farm
- Liste di controllo accessi
- Autorizzazione basata sui tag Device Farm
- Ruoli IAM di Device Farm

Politiche basate sull'identità di Device Farm

Con le policy IAM basate su identità, puoi specificare operazioni e risorse consentite o rifiutate, nonché le condizioni in base alle quali le operazioni sono consentite o rifiutate. Device Farm supporta

azioni, risorse e chiavi di condizione specifiche. Per informazioni su tutti gli elementi utilizzati in una policy JSON, consulta Documentazione di riferimento degli elementi delle policy JSON IAM nella Guida per l'utente IAM.

Azioni

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale entità principale può eseguire operazioni su quali risorse e in quali condizioni.

L'elemento `Action` di una policy JSON descrive le operazioni che è possibile utilizzare per consentire o negare l'accesso in una policy. Includere le operazioni in una policy per concedere le autorizzazioni a eseguire l'operazione associata.

Le azioni politiche in Device Farm utilizzano il seguente prefisso prima dell'azione:`devicefarm:`. Ad esempio, per concedere a qualcuno il permesso di avviare sessioni Selenium con l'operazione dell'`CreateTestGridUrl`API di test del browser desktop Device Farm, includi l'`devicefarm:CreateTestGridUrl`azione nella politica. Le istruzioni della policy devono includere un elemento `Action` o `NotAction`. Device Farm definisce il proprio set di azioni che descrivono le attività che è possibile eseguire con questo servizio.

Per specificare più azioni in una sola istruzione, separa ciascuna di esse con una virgola come mostrato di seguito:

```
"Action": [
      "devicefarm:action1",
      "devicefarm:action2"
```

È possibile specificare più azioni tramite caratteri jolly (*). Ad esempio, per specificare tutte le azioni che iniziano con la parola `List`, includi la seguente azione:

```
"Action": "devicefarm:List*"
```

Per visualizzare un elenco delle azioni di Device Farm, consulta Azioni definite da AWS Device Farm nello IAM Service Authorization Reference.

Resources

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale entità principale può eseguire operazioni su quali risorse e in quali condizioni.

L'elemento JSON `Resource` della policy specifica l'oggetto o gli oggetti ai quali si applica l'operazione. Come best practice, specifica una risorsa utilizzando il suo nome della risorsa Amazon (ARN). Per le azioni che non supportano le autorizzazioni a livello di risorsa, si utilizza un carattere jolly (*) per indicare che l'istruzione si applica a tutte le risorse.

```
"Resource": "*"
```

La risorsa dell'istanza Amazon EC2 ha il seguente ARN:

```
arn:${Partition}:ec2:${Region}:${Account}:instance/${InstanceId}
```

Per ulteriori informazioni sul formato di ARNs, consulta Amazon Resource Names (ARNs) e AWS Service Namespaces.

Ad esempio, per specificare l'istanza `i-1234567890abcdef0` nell'istruzione, utilizza il seguente ARN:

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:instance/i-1234567890abcdef0"
```

Per specificare tutti le istanze che appartengono ad un account specifico, utilizza il carattere jolly (*):

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:instance/*"
```

Alcune azioni di Device Farm, come quelle per la creazione di risorse, non possono essere eseguite su una risorsa. In questi casi, è necessario utilizzare il carattere jolly (*).

```
"Resource": "*"
```

Molte operazioni API di Amazon EC2 coinvolgono più risorse. Ad esempio, `AttachVolume` collega un volume Amazon EBS a un'istanza, pertanto un utente IAM dovrà disporre delle autorizzazioni per utilizzare il volume e l'istanza. Per specificare più risorse in un'unica istruzione, separale ARNs con virgole.

```
"Resource": [
      "resource1",
      "resource2"
```

Per visualizzare un elenco dei tipi di risorse Device Farm e relativi ARNs, consulta Tipi di risorse definiti da AWS Device Farm nello IAM Service Authorization Reference. Per sapere con quali azioni è possibile specificare l'ARN di ogni risorsa, consulta Actions defined by AWS Device Farm nello IAM Service Authorization Reference.

Chiavi di condizione

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale entità principale può eseguire operazioni su quali risorse e in quali condizioni.

L'elemento `Condition` specifica quando le istruzioni vengono eseguite in base a criteri definiti. È possibile compilare espressioni condizionali che utilizzano operatori di condizione, ad esempio uguale a o minore di, per soddisfare la condizione nella policy con i valori nella richiesta. Per visualizzare tutte le chiavi di condizione AWS globali, consulta le chiavi di contesto delle condizioni AWS globali nella Guida per l'utente IAM.

Device Farm definisce il proprio set di chiavi di condizione e supporta anche l'uso di alcune chiavi di condizione globali. Per visualizzare tutte le chiavi di condizione AWS globali, consulta AWS Global Condition Context Keys nella IAM User Guide.

Per visualizzare un elenco delle chiavi di condizione di Device Farm, consulta Condition keys for AWS Device Farm nello IAM Service Authorization Reference. Per sapere con quali azioni e risorse puoi utilizzare una chiave di condizione, consulta Azioni definite da AWS Device Farm nello IAM Service Authorization Reference.

Esempi

Per visualizzare esempi di policy basate sull'identità di Device Farm, vedere. Esempi di policy basate sull'identità di AWS Device Farm

Politiche basate sulle risorse di Device Farm

Device Farm non supporta politiche basate sulle risorse.

Liste di controllo accessi

Device Farm non supporta gli elenchi di controllo degli accessi (ACLs).

Autorizzazione basata sui tag Device Farm

È possibile allegare tag alle risorse di Device Farm o passare i tag in una richiesta a Device Farm. Per controllare l'accesso basato su tag, fornire informazioni sui tag nell'elemento condizione di una policy utilizzando le chiavi di condizione `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` o `aws:TagKeys`. Per ulteriori informazioni sull'etichettatura delle risorse di Device Farm, vedereInserimento di tag in Device Farm.

Per visualizzare una policy basata sulle identità di esempio per limitare l'accesso a una risorsa basata su tag su tale risorsa, consulta Visualizzazione dei progetti di test del browser desktop Device Farm in base ai tag.

Ruoli IAM di Device Farm

Un ruolo IAM è un'entità nel tuo AWS account che dispone di autorizzazioni specifiche.

Utilizzo di credenziali temporanee con Device Farm

Device Farm supporta l'uso di credenziali temporanee.

È possibile utilizzare credenziali temporanee per accedere con la federazione e assumere un ruolo IAM o un ruolo tra account. Puoi ottenere credenziali di sicurezza temporanee chiamando operazioni AWS STS API come o. AssumeRoleGetFederationToken

Ruoli collegati ai servizi

I ruoli collegati ai AWS servizi consentono ai servizi di accedere alle risorse di altri servizi per completare un'azione per conto dell'utente. I ruoli collegati ai servizi sono visualizzati nell'account IAM e sono di proprietà del servizio. Un amministratore IAM può visualizzare, ma non modificare, le autorizzazioni per i ruoli collegati ai servizi.

Device Farm utilizza ruoli collegati ai servizi nella funzionalità di test del browser desktop Device Farm. Per informazioni su questi ruoli, consulta Using Service-Linked Roles in Device Farm Desktop Browser Testing nella guida per sviluppatori.

Ruoli di servizio

Device Farm non supporta i ruoli di servizio.

Questa funzionalità consente a un servizio di assumere un ruolo di servizio per conto dell'utente. Questo ruolo consente al servizio di accedere alle risorse in altri servizi per completare un'azione per conto dell'utente. I ruoli dei servizi sono visualizzati nell'account IAM e sono di proprietà dell'account.

Ciò significa che un amministratore IAM può modificare le autorizzazioni per questo ruolo. Tuttavia, questo potrebbe pregiudicare la funzionalità del servizio.

Gestione dell'accesso tramite policy

Puoi controllare l'accesso AWS creando policy e associandole a AWS identità o risorse. Una policy definisce le autorizzazioni quando è associata a un'identità o a una risorsa. AWS valuta queste politiche quando un preside effettua una richiesta. La maggior parte delle politiche viene archiviata AWS come documenti JSON. Per maggiori informazioni sui documenti delle policy JSON, consulta Panoramica delle policy JSON nella Guida per l'utente IAM.

Utilizzando le policy, gli amministratori specificano chi ha accesso a cosa definendo quale principale può eseguire azioni su quali risorse e in quali condizioni.

Per impostazione predefinita, utenti e ruoli non dispongono di autorizzazioni. Un amministratore IAM crea le policy IAM e le aggiunge ai ruoli, che gli utenti possono quindi assumere. Le policy IAM definiscono le autorizzazioni indipendentemente dal metodo utilizzato per eseguirle.

Policy basate sull'identità

Le policy basate su identità sono documenti di policy di autorizzazione JSON che è possibile collegare a un'identità (utente, gruppo o ruolo). Tali policy controllano le operazioni autorizzate per l'identità, nonché le risorse e le condizioni in cui possono essere eseguite. Per informazioni su come creare una policy basata su identità, consultare Definizione di autorizzazioni personalizzate IAM con policy gestite dal cliente nella Guida per l'utente IAM.

Le policy basate su identità possono essere policy in linea (con embedding direttamente in una singola identità) o policy gestite (policy autonome collegate a più identità). Per informazioni su come scegliere tra una policy gestita o una policy inline, consulta Scegliere tra policy gestite e policy in linea nella Guida per l'utente di IAM.

La tabella seguente descrive le policy gestite da Device Farm in AWS.

Modifica	Descrizione	Data
AWSDeviceFarmFullAccess	Fornisce accesso completo a tutte le operazioni di AWS Device Farm.	15 luglio 2015

Modifica	Descrizione	Data
AWSServiceRoleForDeviceFarmTestGrid	Consente a Device Farm di accedere alle risorse AWS per tuo conto.	20 maggio 2021

Altri tipi di policy

AWS supporta tipi di policy aggiuntivi che possono impostare le autorizzazioni massime concesse dai tipi di policy più comuni:

- Limiti delle autorizzazioni: imposta il numero massimo di autorizzazioni che una policy basata su identità ha la possibilità di concedere a un'entità IAM. Per ulteriori informazioni, consulta Limiti delle autorizzazioni per le entità IAM nella Guida per l'utente di IAM.
- Politiche di controllo del servizio (SCPs): specificano le autorizzazioni massime per un'organizzazione o un'unità organizzativa in. AWS Organizations Per ulteriori informazioni, consultare Policy di controllo dei servizi nella Guida per l'utente di AWS Organizations .
- Politiche di controllo delle risorse (RCPs): imposta le autorizzazioni massime disponibili per le risorse nei tuoi account. Per ulteriori informazioni, consulta Politiche di controllo delle risorse (RCPs) nella Guida per l'AWS Organizations utente.
- Policy di sessione: policy avanzate passate come parametro quando si crea una sessione temporanea per un ruolo o un utente federato. Per maggiori informazioni, consultare Policy di sessione nella Guida per l'utente IAM.

Più tipi di policy

Quando a una richiesta si applicano più tipi di policy, le autorizzazioni risultanti sono più complicate da comprendere. Per scoprire come si AWS determina se consentire o meno una richiesta quando sono coinvolti più tipi di policy, consulta Logica di valutazione delle policy nella IAM User Guide.

Esempi di policy basate sull'identità di AWS Device Farm

Per impostazione predefinita, gli utenti e i ruoli IAM non dispongono dell'autorizzazione per creare o modificare risorse Device Farm. Inoltre, non possono eseguire attività utilizzando l' AWS API Console di gestione AWS AWS CLI, o. Un amministratore IAM deve creare policy IAM che concedono a utenti e ruoli l'autorizzazione per eseguire operazioni API specifiche sulle risorse specificate di cui

hanno bisogno. L'amministratore deve quindi allegare queste policy a utenti o IAM che richiedono tali autorizzazioni.

Per informazioni su come creare una policy basata su identità IAM utilizzando questi documenti di policy JSON di esempio, consulta Creazione di policy nella scheda JSON nella Guida per l'utente IAM.

Argomenti

- Best practice delle policy
- Consentire agli utenti di visualizzare le loro autorizzazioni
- Accesso a un progetto di test del browser desktop Device Farm
- Visualizzazione dei progetti di test del browser desktop Device Farm in base ai tag

Best practice delle policy

Le politiche basate sull'identità determinano se qualcuno può creare, accedere o eliminare le risorse Device Farm nel tuo account. Queste azioni possono comportare costi aggiuntivi per l' Account AWS. Quando si creano o modificano policy basate sull'identità, seguire queste linee guida e raccomandazioni:

- Inizia con le policy AWS gestite e passa alle autorizzazioni con privilegi minimi: per iniziare a concedere autorizzazioni a utenti e carichi di lavoro, utilizza le politiche gestite che concedono le autorizzazioni per molti casi d'uso comuni.AWS Sono disponibili nel tuo. Account AWS Ti consigliamo di ridurre ulteriormente le autorizzazioni definendo politiche gestite dai AWS clienti specifiche per i tuoi casi d'uso. Per maggiori informazioni, consulta Policy gestite da AWS o Policy gestite da AWS per le funzioni dei processi nella Guida per l'utente di IAM.
- Applicazione delle autorizzazioni con privilegio minimo - Quando si impostano le autorizzazioni con le policy IAM, concedere solo le autorizzazioni richieste per eseguire un'attività. È possibile farlo definendo le azioni che possono essere intraprese su risorse specifiche in condizioni specifiche, note anche come autorizzazioni con privilegio minimo. Per maggiori informazioni sull'utilizzo di IAM per applicare le autorizzazioni, consulta Policy e autorizzazioni in IAM nella Guida per l'utente di IAM.
- Condizioni d'uso nelle policy IAM per limitare ulteriormente l'accesso - Per limitare l'accesso ad azioni e risorse è possibile aggiungere una condizione alle policy. Ad esempio, è possibile scrivere una condizione di policy per specificare che tutte le richieste devono essere inviate utilizzando SSL. Puoi anche utilizzare le condizioni per concedere l'accesso alle azioni del servizio

se vengono utilizzate tramite uno specifico Servizio AWS, ad esempio CloudFormation. Per maggiori informazioni, consultare la sezione Elementi delle policy JSON di IAM: condizione nella Guida per l'utente di IAM.

- Utilizzo dello strumento di analisi degli accessi IAM per convalidare le policy IAM e garantire autorizzazioni sicure e funzionali - Lo strumento di analisi degli accessi IAM convalida le policy nuove ed esistenti in modo che aderiscano al linguaggio (JSON) della policy IAM e alle best practice di IAM. Lo strumento di analisi degli accessi IAM offre oltre 100 controlli delle policy e consigli utili per creare policy sicure e funzionali. Per maggiori informazioni, consultare Convalida delle policy per il Sistema di analisi degli accessi IAM nella Guida per l'utente di IAM.
- Richiedi l'autenticazione a più fattori (MFA): se hai uno scenario che richiede utenti IAM o un utente root nel Account AWS tuo, attiva l'MFA per una maggiore sicurezza. Per richiedere la MFA quando vengono chiamate le operazioni API, aggiungere le condizioni MFA alle policy. Per maggiori informazioni, consultare Protezione dell'accesso API con MFA nella Guida per l'utente di IAM.

Per maggiori informazioni sulle best practice in IAM, consulta Best practice di sicurezza in IAM nella Guida per l'utente di IAM.

Consentire agli utenti di visualizzare le loro autorizzazioni

Questo esempio mostra in che modo è possibile creare una policy che consente agli utenti IAM di visualizzare le policy inline e gestite che sono collegate alla relativa identità utente. Questa policy include le autorizzazioni per completare questa azione sulla console o utilizzando programmaticamente l'API o. 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": "*"
        }
    ]
}
```

Accesso a un progetto di test del browser desktop Device Farm

In questo esempio, vuoi concedere a un utente IAM del tuo AWS account l'accesso a uno dei tuoi progetti di test del browser desktop Device Farm,. `arn:aws:devicefarm:us-west-2:111122223333:testgrid-project:123e4567-e89b-12d3-a456-426655441111` Si desidera che l'account sia in grado di visualizzare gli elementi correlati al progetto.

Oltre all'endpoint `devicefarm:GetTestGridProject`, l'account deve avere gli endpoint `devicefarm:ListTestGridSessions`, `devicefarm:GetTestGridSession`, `devicefarm:ListTestGridSessionActions` e `devicefarm:ListTestGridSessionArtifacts`.

Se si utilizzano sistemi CI, è necessario fornire a ciascuna esecuzione CI credenziali di accesso univoche. Ad esempio, è improbabile che un sistema CI abbia bisogno di più autorizzazioni rispetto a `devicefarm:ScheduleRun` o `devicefarm:CreateUpload`. La seguente policy IAM delinea una policy minima per consentire a un CI runner di avviare un test di un nuovo test dell'app nativa di Device Farm creando un caricamento e utilizzandolo per pianificare un'esecuzione di test:

Visualizzazione dei progetti di test del browser desktop Device Farm in base ai tag

È possibile utilizzare le condizioni della policy basata sull'identità per controllare l'accesso alle risorse di Device Farm in base ai tag. In questo esempio viene illustrato come creare una policy che

consenta la visualizzazione di un progetto e delle relative sessioni. L'autorizzazione viene concessa se il tag `0wner` della risorsa richiesta corrisponde al nome utente dell'account richiedente.

Puoi allegare questa policy agli utenti IAM nel tuo account. Se un utente denominato `richard-roe` tenta di visualizzare un progetto o una sessione di Device Farm, il progetto deve essere taggato `0wner=richard-roe` o`owner=richard-roe`. In caso contrario, a questo utente viene negato l'accesso. La chiave di tag di condizione `0wner` corrisponde sia a `0wner` che a `owner` perché i nomi delle chiavi di condizione non distinguono tra maiuscole e minuscole. Per ulteriori informazioni, consulta la sezione Elementi delle policy JSON di IAM: condizione nella Guida per l'utente di IAM.

Risoluzione dei problemi di identità e accesso ad AWS Device Farm

Utilizza le seguenti informazioni per aiutarti a diagnosticare e risolvere i problemi più comuni che potresti riscontrare quando lavori con Device Farm e IAM.

Non sono autorizzato a eseguire un'azione in Device Farm

Se ricevi un messaggio di errore Console di gestione AWS che indica che non sei autorizzato a eseguire un'azione, devi contattare l'amministratore per ricevere assistenza. L'amministratore è la persona che ti ha fornito il nome utente e la password.

Il seguente errore di esempio si verifica quando l'utente IAM tenta di utilizzare la console per visualizzare i dettagli di un'esecuzione, ma non dispone `devicefarm:GetRun` delle autorizzazioni. `mateojackson`

```
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 questo caso, Mateo chiede al suo amministratore di aggiornare le sue policy per poter accedere a `devicefarm:GetRun` sulla risorsa `arn:aws:devicefarm:us-west-2:123456789101:run:123e4567-e89b-12d3-a456-426655440000/123e4567-e89b-12d3-a456-426655441111` mediante l'operazione `devicefarm:GetRun`.

Non sono autorizzato a eseguire iam: PassRole

Se ricevi un messaggio di errore indicante che non sei autorizzato a eseguire l'`iam:PassRole`azione, le tue politiche devono essere aggiornate per consentirti di trasferire un ruolo a Device Farm.

Alcuni Servizi AWS consentono di trasferire un ruolo esistente a quel servizio invece di creare un nuovo ruolo di servizio o un ruolo collegato al servizio. Per eseguire questa operazione, è necessario disporre delle autorizzazioni per trasmettere il ruolo al servizio.

Il seguente errore di esempio si verifica quando un utente IAM denominato `marymajor` tenta di utilizzare la console per eseguire un'azione in Device Farm. Tuttavia, l'azione richiede che il servizio disponga delle autorizzazioni concesse da un ruolo di servizio. Mary non dispone delle autorizzazioni per trasmettere il ruolo al servizio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In questo caso, le policy di Mary devono essere aggiornate per poter eseguire l'operazione `iam:PassRole`.

Se hai bisogno di aiuto, contatta il tuo AWS amministratore. L'amministratore è colui che ti ha fornito le credenziali di accesso.

Desidero visualizzare le mie chiavi di accesso

Dopo aver creato le chiavi di accesso utente IAM, è possibile visualizzare il proprio ID chiave di accesso in qualsiasi momento. Tuttavia, non è possibile visualizzare nuovamente la chiave di accesso segreta. Se perdi la chiave segreta, dovrai creare una nuova coppia di chiavi di accesso.

Le chiavi di accesso sono composte da due parti: un ID chiave di accesso (ad esempio `AKIAIOSFODNN7EXAMPLE`) e una chiave di accesso segreta (ad esempio, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Come un nome utente e una password, è necessario utilizzare sia l'ID chiave di accesso sia la chiave di accesso segreta insieme per autenticare le richieste dell'utente. Gestisci le tue chiavi di accesso in modo sicuro mentre crei il nome utente e la password.

> **Important**
>
> Non fornire le chiavi di accesso a terze parti, neppure per aiutare a trovare l'ID utente canonico. In questo modo, potresti concedere a qualcuno l'accesso permanente al tuo Account AWS.

Quando crei una coppia di chiavi di accesso, ti viene chiesto di salvare l'ID chiave di accesso e la chiave di accesso segreta in una posizione sicura. La chiave di accesso segreta è disponibile solo al momento della creazione. Se si perde la chiave di accesso segreta, è necessario aggiungere nuove

chiavi di accesso all'utente IAM. È possibile avere massimo due chiavi di accesso. Se se ne hanno già due, è necessario eliminare una coppia di chiavi prima di crearne una nuova. Per visualizzare le istruzioni, consulta Gestione delle chiavi di accesso nella Guida per l'utente di IAM.

Sono un amministratore e desidero consentire ad altri di accedere a Device Farm

Per consentire ad altri di accedere a Device Farm, è necessario concedere l'autorizzazione alle persone o alle applicazioni che devono accedere. Se si utilizza AWS IAM Identity Center per gestire persone e applicazioni, si assegnano set di autorizzazioni a utenti o gruppi per definire il loro livello di accesso. I set di autorizzazioni creano e assegnano automaticamente le policy IAM ai ruoli IAM associati alla persona o all'applicazione. Per ulteriori informazioni, consulta Set di autorizzazioni nella Guida per l'AWS IAM Identity Center utente.

Se non utilizzi IAM Identity Center, devi creare entità IAM (utenti o ruoli) per le persone o le applicazioni che necessitano di accesso. È quindi necessario allegare una politica all'entità che concede loro le autorizzazioni corrette in Device Farm. Dopo aver concesso le autorizzazioni, fornite le credenziali all'utente o allo sviluppatore dell'applicazione. Utilizzeranno tali credenziali per accedere. AWSPer ulteriori informazioni sulla creazione di utenti, gruppi, policy e autorizzazioni IAM, consulta IAM Identities and Policies and permissions in IAM nella IAM User Guide.

Desidero consentire a persone esterne al mio AWS account di accedere alle mie risorse di Device Farm

È possibile creare un ruolo con il quale utenti in altri account o persone esterne all'organizzazione possono accedere alle tue risorse. È possibile specificare chi è attendibile per l'assunzione del ruolo. Per i servizi che supportano politiche basate sulle risorse o liste di controllo degli accessi (ACLs), puoi utilizzare tali politiche per concedere alle persone l'accesso alle tue risorse.

Per maggiori informazioni, consulta gli argomenti seguenti:

- Per sapere se Device Farm supporta queste funzionalità, vedereCome funziona AWS Device Farm con IAM.
- Per scoprire come fornire l'accesso alle tue risorse su tutto Account AWS ciò che possiedi, consulta Fornire l'accesso a un utente IAM in un altro Account AWS di tua proprietà nella IAM User Guide.
- Per scoprire come fornire l'accesso alle tue risorse a terze parti Account AWS, consulta Fornire l'accesso a soggetti Account AWS di proprietà di terze parti nella Guida per l'utente IAM.
- Per informazioni su come fornire l'accesso tramite la federazione delle identità, consulta Fornire l'accesso a utenti autenticati esternamente (federazione delle identità) nella Guida per l'utente IAM.

- Per informazioni sulle differenze di utilizzo tra ruoli e policy basate su risorse per l'accesso multi-account, consulta Accesso a risorse multi-account in IAM nella Guida per l'utente IAM.

Convalida della conformità per AWS Device Farm

I revisori esterni valutano la sicurezza e la conformità nell' AWS Device Farm ambito di più programmi di AWS conformità. Questi includono SOC, PCI, FedRAMP, HIPAA e altri. AWS Device Farm non rientra nell'ambito di alcun programma di conformità. AWS

Per un elenco di AWS servizi nell'ambito di programmi di conformità specifici, consulta Servizi AWS nell'ambito del programma di conformità . Per informazioni generali, consulta Programmi di AWS conformità Programmi di di .

È possibile scaricare report di audit di terze parti utilizzando AWS Artifact. Per ulteriori informazioni, consulta Download dei report in AWS Artifact.

La responsabilità di conformità dell'utente nell'utilizzo di Device Farm è determinata dalla sensibilità dei dati, dagli obiettivi di conformità dell'azienda e dalle leggi e dai regolamenti applicabili. AWS fornisce le seguenti risorse per contribuire alla conformità:

- Security and Compliance Quick Start Guides (Guide Quick Start Sicurezza e compliance): queste guide alla distribuzione illustrano considerazioni relative all'architettura e forniscono procedure per la distribuzione di ambienti di base incentrati sulla sicurezza e sulla conformità su AWS.
- AWS Risorse per la per la conformità: questa raccolta di cartelle di lavoro e guide potrebbe riguardare il settore e la località in cui operi.
- Evaluating Resources with Rules nella AWS Config Developer Guide: AWS Config valuta la conformità delle configurazioni delle risorse alle pratiche interne, alle linee guida del settore e alle normative.
- AWS Security Hub CSPM— Questo AWS servizio offre una visione completa dello stato di sicurezza dell'utente e consente di verificare la conformità agli standard e alle best practice del settore della sicurezza. AWS

Protezione dei dati in AWS Device Farm

Il modello di responsabilità AWS condivisa modello di si applica alla protezione dei dati in AWS Device Farm (Device Farm). Come descritto in questo modello, AWS è responsabile della protezione dell'infrastruttura globale che gestisce tutti i Cloud AWS. L'utente è responsabile del controllo

dei contenuti ospitati su questa infrastruttura. L'utente è inoltre responsabile della configurazione della protezione e delle attività di gestione per i Servizi AWS utilizzati. Per maggiori informazioni sulla privacy dei dati, consulta le Domande frequenti sulla privacy dei dati. Per informazioni sulla protezione dei dati in Europa, consulta il post del blog relativo al AWS Modello di responsabilità condivisa e GDPR nel AWS Blog sulla sicurezza.

Ai fini della protezione dei dati, consigliamo di proteggere Account AWS le credenziali e configurare i singoli utenti con AWS IAM Identity Center or AWS Identity and Access Management (IAM). In tal modo, a ogni utente verranno assegnate solo le autorizzazioni necessarie per svolgere i suoi compiti. Suggeriamo, inoltre, di proteggere i dati nei seguenti modi:

- Utilizza l'autenticazione a più fattori (MFA) con ogni account.
- SSL/TLS Da utilizzare per comunicare con AWS le risorse. È richiesto TLS 1.2 ed è consigliato TLS 1.3.
- Configura l'API e la registrazione delle attività degli utenti con AWS CloudTrail. Per informazioni sull'utilizzo dei CloudTrail percorsi per acquisire AWS le attività, consulta Lavorare con i CloudTrail percorsi nella Guida per l'AWS CloudTrail utente.
- Utilizza soluzioni di AWS crittografia, insieme a tutti i controlli di sicurezza predefiniti all'interno Servizi AWS.
- Utilizza i servizi di sicurezza gestiti avanzati, come Amazon Macie, che aiutano a individuare e proteggere i dati sensibili archiviati in Amazon S3.
- Se hai bisogno di moduli crittografici convalidati FIPS 140-3 per accedere AWS tramite un'interfaccia a riga di comando o un'API, usa un endpoint FIPS. Per ulteriori informazioni sugli endpoint FIPS disponibili, consulta il Federal Information Processing Standard (FIPS) 140-3.

Ti consigliamo di non inserire mai informazioni riservate o sensibili, ad esempio gli indirizzi e-mail dei clienti, nei tag o nei campi di testo in formato libero, ad esempio nel campo Nome. Ciò include quando lavori con Device Farm o altro Servizi AWS utilizzando la console AWS CLI, l'API o AWS SDKs. I dati inseriti nei tag o nei campi di testo in formato libero utilizzati per i nomi possono essere utilizzati per i la fatturazione o i log di diagnostica. Quando si fornisce un URL a un server esterno, suggeriamo vivamente di non includere informazioni sulle credenziali nell'URL per convalidare la richiesta al server.

Crittografia dei dati in transito

Gli endpoint Device Farm supportano solo HTTPS firmato (SSL/TLS) requests except where otherwise noted. All content retrieved from or placed in Amazon S3 through upload URLs is encrypted

using SSL/TLS. Per ulteriori informazioni su come vengono effettuate le richieste HTTPS AWS, consulta Firmare le richieste AWS API nella Guida AWS generale.

La crittografia e la protezione di tutte le comunicazioni effettuate dalle applicazioni testate, nonché di eventuali applicazioni aggiuntive installate durante l'esecuzione di test sul dispositivo, è responsabilità dell'utente.

Crittografia dei dati a riposo

La funzionalità di test del browser desktop di Device Farm supporta la crittografia a riposo per gli artefatti generati durante i test.

I dati di test fisici dei dispositivi mobili di Device Farm non sono crittografati quando sono inattivi.

Conservazione dei dati

I dati in Device Farm vengono conservati per un periodo di tempo limitato. Dopo la scadenza del periodo di conservazione, i dati vengono rimossi dall'archivio di backup di Device Farm.

Tipo di contenuto	Periodo di conservazione (giorni)	Periodo di conservazione dei metadati (giorni)
Applicazioni caricate	30	30
Pacchetti di test caricati	30	30
Log	400	400
Registrazioni video e altri artefatti	400	400

È responsabilità dell'utente archiviare qualsiasi contenuto che desidera conservare per periodi più lunghi.

Gestione dei dati

I dati in Device Farm vengono gestiti in modo diverso a seconda delle funzionalità utilizzate. Questa sezione spiega come vengono gestiti i dati durante e dopo l'utilizzo di Device Farm.

Test del browser desktop

Le istanze utilizzate durante le sessioni Selenium non vengono salvate. Tutti i dati generati come risultato delle interazioni del browser vengono scartati al termine della sessione.

Questa funzionalità attualmente supporta la crittografia a riposo per gli artefatti generati durante il test.

Test fisici dei dispositivi

Le seguenti sezioni forniscono informazioni sui passaggi AWS necessari per pulire o distruggere i dispositivi dopo aver utilizzato Device Farm.

I dati di test sui dispositivi mobili fisici di Device Farm non sono crittografati quando sono inattivi.

Flotte di dispositivi pubblici

Una volta completata l'esecuzione del test, Device Farm esegue una serie di attività di pulizia su ogni dispositivo del parco dispositivi pubblici, inclusa la disinstallazione dell'app. Se non siamo in grado di confermare la disinstallazione della tua app o qualsiasi altra fase di pulizia, il dispositivo viene reimpostato ai valori di fabbrica prima di essere riutilizzato.

> **Note**
>
> In alcuni casi è possibile che i dati persistano tra una sessione e l'altra, soprattutto se si utilizza il sistema del dispositivo al di fuori del contesto dell'app. Per questo motivo, e poiché Device Farm acquisisce video e registri delle attività che si svolgono durante l'utilizzo di ciascun dispositivo, consigliamo di non inserire informazioni sensibili (ad esempio, account Google o ID Apple), informazioni personali e altri dettagli sensibili alla sicurezza durante le sessioni di test automatizzate e di accesso remoto.

Dispositivi privati

Dopo la scadenza o la risoluzione del contratto relativo a un dispositivo privato, quest'ultimo viene ritirato dall'uso ed eliminato in modo sicuro in conformità con le policy di eliminazione di AWS. Per ulteriori informazioni, consulta Dispositivi privati in AWS Device Farm.

Gestione delle chiavi

Attualmente, Device Farm non offre alcuna gestione di chiavi esterne per la crittografia dei dati, a riposo o in transito.

Riservatezza del traffico Internet

Device Farm può essere configurato, solo per dispositivi privati, per utilizzare gli endpoint Amazon VPC per connettersi alle tue risorse. AWS L'accesso a qualsiasi AWS infrastruttura non pubblica associata al tuo account (ad esempio, EC2 istanze Amazon senza un indirizzo IP pubblico) deve utilizzare un endpoint Amazon VPC. Indipendentemente dalla configurazione degli endpoint VPC, Device Farm isola il traffico dagli altri utenti in tutta la rete Device Farm.

Non è garantito che le connessioni all'esterno della AWS rete siano protette o protette, ed è responsabilità dell'utente proteggere tutte le connessioni Internet create dalle applicazioni.

Resilienza in AWS Device Farm

L'infrastruttura AWS globale è costruita attorno a AWS regioni e zone di disponibilità. AWS Le regioni forniscono più zone di disponibilità fisicamente separate e isolate, collegate con reti a bassa latenza, ad alto throughput e altamente ridondanti. Con le zone di disponibilità, puoi progettare e gestire applicazioni e database che eseguono automaticamente il failover tra zone di disponibilità senza interruzioni. Le zone di disponibilità sono più disponibili, tolleranti ai guasti e scalabili rispetto alle infrastrutture a data center singolo o multiplo tradizionali.

Per ulteriori informazioni su AWS regioni e zone di disponibilità, consulta Global Infrastructure.AWS

Poiché Device Farm è disponibile solo nella `us-west-2` regione, si consiglia vivamente di implementare processi di backup e ripristino. Device Farm non dovrebbe essere l'unica fonte di contenuti caricati.

Device Farm non garantisce la disponibilità di dispositivi pubblici. Questi dispositivi vengono inseriti e rimossi dal pool di dispositivi pubblici in base a una varietà di fattori, come il tasso di errore e lo stato di quarantena. Si consiglia di non dipendere dalla disponibilità di un dispositivo nel pool di dispositivi pubblici.

Sicurezza dell'infrastruttura in AWS Device Farm

In quanto servizio gestito, AWS Device Farm è protetto dalla sicurezza della rete AWS globale. Per informazioni sui servizi AWS di sicurezza e su come AWS protegge l'infrastruttura, consulta AWS Cloud Security. Per progettare il tuo AWS ambiente utilizzando le migliori pratiche per la sicurezza dell'infrastruttura, vedi Infrastructure Protection in Security Pillar AWS Well-Architected Framework.

Si utilizzano chiamate API AWS pubblicate per accedere a Device Farm attraverso la rete. I client devono supportare quanto segue:

- Transport Layer Security (TLS). È richiesto TLS 1.2 ed è consigliato TLS 1.3.
- Suite di cifratura con Perfect Forward Secrecy (PFS), ad esempio Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). La maggior parte dei sistemi moderni, come Java 7 e versioni successive, supporta tali modalità.

Inoltre, le richieste devono essere firmate utilizzando un ID chiave di accesso e una chiave di accesso segreta associata a un principale IAM. In alternativa è possibile utilizzare AWS Security Token Service (AWS STS) per generare credenziali di sicurezza temporanee per sottoscrivere le richieste.

Sicurezza dell'infrastruttura per il test dei dispositivi fisici

I dispositivi sono fisicamente separati durante il test fisico del dispositivo. L'isolamento della rete impedisce la comunicazione tra dispositivi su reti wireless.

I dispositivi pubblici sono condivisi e Device Farm fa del suo meglio per proteggere i dispositivi nel tempo. Alcune azioni, ad esempio i tentativi di acquisire diritti di amministratore completi su un dispositivo (una pratica denominata rooting o jailbreak) causano la messa in quarantena dei dispositivi pubblici. Vengono rimossi automaticamente dal pool pubblico e inseriti in revisione manuale.

I dispositivi privati sono accessibili solo da AWS account esplicitamente autorizzati a farlo. Device Farm isola fisicamente questi dispositivi dagli altri dispositivi e li mantiene su una rete separata.

Sui dispositivi gestiti privatamente, i test possono essere configurati per utilizzare un endpoint Amazon VPC per proteggere le connessioni in entrata e in uscita dal tuo account. AWS

Sicurezza dell'infrastruttura per il test dei browser desktop

Quando si utilizza la caratteristica di test del browser desktop, tutte le sessioni di test sono separate l'una dall'altra. Le istanze Selenium non possono comunicare tra loro senza una terza parte intermedia, esterna a. AWS

Tutto il traffico verso i WebDriver controller Selenium deve essere effettuato tramite l'endpoint HTTPS generato con. `createTestGridUrl`

L'utente ha la responsabilità di assicurarsi che ogni istanza di test di Device Farm abbia accesso sicuro alle risorse da essa testate. Per impostazione predefinita, le istanze di test dei browser desktop di Device Farm hanno accesso alla rete Internet pubblica. Quando colleghi un'istanza a un VPC, si comporta come qualsiasi altra istanza EC2, con accesso alle risorse determinato dalla configurazione del VPC e dai componenti di rete associati. AWS fornisce gruppi di sicurezza e liste di controllo degli accessi di rete (ACLs) per aumentare la sicurezza nel tuo VPC. I gruppi di sicurezza controllano il traffico in entrata e in uscita per le tue risorse e la rete ACLs controlla il traffico in entrata e in uscita per le tue sottoreti. I gruppi di sicurezza forniscono un controllo di accesso sufficiente per la maggior parte delle sottoreti. Puoi usare la rete ACLs se desideri un ulteriore livello di sicurezza per il tuo VPC. Per linee guida generali sulle best practice di sicurezza per l'utilizzo di Amazon VPCs, consulta le best practice di sicurezza per il tuo VPC nella Amazon Virtual Private Cloud User Guide.

Analisi e gestione delle vulnerabilità di configurazione in Device Farm

Device Farm consente di eseguire software che non sono attivamente gestiti o aggiornati dal fornitore, ad esempio il fornitore del sistema operativo, il fornitore dell'hardware o l'operatore telefonico. Device Farm fa del suo meglio per mantenere aggiornato il software, ma non garantisce che una particolare versione del software su un dispositivo fisico sia aggiornata, poiché la progettazione consente l'utilizzo di software potenzialmente vulnerabili.

Ad esempio, se viene eseguito un test su un dispositivo con Android 4.4.2, Device Farm non garantisce che il dispositivo sia aggiornato contro la vulnerabilità di Android nota come. StageFright Spetta al fornitore (e talvolta al gestore) del dispositivo fornire aggiornamenti per la sicurezza ai dispositivi. Non è possibile garantire che un'applicazione dannosa che utilizza questa vulnerabilità sia catturata dalla quarantena automatica.

I dispositivi privati vengono mantenuti in base al contratto stipulato con. AWS

Device Farm compie ogni sforzo possibile per impedire alle applicazioni dei clienti di eseguire azioni come il rooting o il jailbreak. Device Farm rimuove i dispositivi messi in quarantena dal pool pubblico fino a quando non vengono esaminati manualmente.

È tua responsabilità mantenere aggiornate tutte le librerie o le versioni del software che usi nei tuoi test, come Python wheels e Ruby gems. Device Farm consiglia di aggiornare le librerie di test.

Queste risorse consentono di mantenere aggiornate le dipendenze dei test:

- Per informazioni su come proteggere le gemme Ruby, consulta Security Practices sul RubyGems sito web.
- Per informazioni sul pacchetto di sicurezza utilizzato da Pipenv e approvato dalla Python Packaging Authority per scansionare il grafico delle dipendenze alla ricerca di vulnerabilità note, vedere Rilevamento delle vulnerabilità di sicurezza su. GitHub
- Per informazioni sul correttore di dipendenze Maven dell'Open Web Application Security Project (OWASP), consulta OWASP sul sito Web OWASP. DependencyCheck

È importante ricordare che anche se un sistema automatizzato non ritiene che ci siano problemi di sicurezza noti, ciò non significa che non ci siano problemi di sicurezza. Utilizzare sempre la due diligence quando si utilizzano librerie o strumenti di terze parti e verificare le firme crittografiche quando possibile o ritenuto necessario.

Risposta agli incidenti in Device Farm

Device Farm monitora continuamente i dispositivi per rilevare comportamenti che potrebbero indicare problemi di sicurezza. Se AWS viene a conoscenza di un caso in cui i dati del cliente, come i risultati dei test o i file scritti su un dispositivo pubblico, sono accessibili da un altro cliente, AWS contatta i clienti interessati, in base alle politiche standard di avviso e segnalazione degli incidenti utilizzate in tutti i servizi. AWS

Registrazione e monitoraggio in Device Farm

Questo servizio supporta AWS CloudTrail, che è un servizio che registra AWS le chiamate Account AWS e invia i file di registro a un bucket Amazon S3. Utilizzando le informazioni raccolte da CloudTrail, è possibile determinare a quali richieste sono state inoltrate correttamente Servizi AWS, chi ha effettuato la richiesta, quando è stata effettuata e così via. Per ulteriori informazioni CloudTrail, incluso come attivarlo e trovare i file di registro, consulta la Guida per l'AWS CloudTrail utente.

Per informazioni sull'utilizzo CloudTrail con Device Farm, vedereRegistrazione delle chiamate API AWS Device Farm con AWS CloudTrail.

Best practice di sicurezza per Device Farm

Device Farm offre una serie di funzionalità di sicurezza da prendere in considerazione durante lo sviluppo e l'implementazione delle proprie politiche di sicurezza. Le seguenti best practice sono linee guida generali e non rappresentano una soluzione di sicurezza completa. Poiché queste best practice potrebbero non essere appropriate o sufficienti per l'ambiente, sono da considerare come considerazioni utili anziché prescrizioni.

- Concedere a qualsiasi sistema di integrazione continua (CI) utilizzato i privilegi minimi possibili in IAM. Prendere in considerazione l'utilizzo di credenziali temporanee per ogni test di sistema CI in modo che, anche se un sistema CI è compromesso, non può effettuare richieste false. Per ulteriori informazioni sulle credenziali temporanee, consulta la IAM User Guide.
- Utilizzare i comandi `adb` in un ambiente di test personalizzato per pulire qualsiasi contenuto creato dall'applicazione. Per ulteriori informazioni sugli ambienti di test personalizzati, consulta Ambienti di test personalizzati

Limiti in AWS Device Farm

Argomenti

Limiti del servizio

- Non vi è alcun limite al numero di dispositivi che puoi includere in un'esecuzione di test. Tuttavia, il numero massimo di dispositivi che Device Farm testerà contemporaneamente durante un test è cinque. Questo numero può essere aumentato su richiesta e valutato caso per caso dal team di assistenza.
- Non vi è alcun limite al numero di esecuzioni che puoi pianificare. Tieni presente che possono rimanere in coda solo per un massimo di 24 ore.
- Esiste un limite rigido di 150 minuti per la durata di una sessione di accesso remoto.
- Esiste un limite rigido di 150 minuti alla durata di un test automatico
- Il numero massimo di lavori in corso, compresi i lavori in coda in sospeso sul tuo account, è 250. Si tratta di un limite flessibile.
- Non c'è limite al numero di dispositivi che puoi includere in un'esecuzione di test. Il numero di dispositivi (job) che possono eseguire i test in parallelo in un dato momento è pari alla concorrenza a livello di account. La concorrenza predefinita a livello di account per l'uso misurato in Device Farm è cinque.
- Il limite di concorrenza misurato può essere aumentato su richiesta fino a una determinata soglia a seconda del caso d'uso. La concorrenza predefinita a livello di account per un utilizzo illimitato è pari al numero di slot a cui sei abbonato per quella piattaforma.

  Per ulteriori informazioni sui limiti di concorrenza misurati di default o sulle quote in generale, consulta la pagina Quote.
- Un'esecuzione di automazione che non utilizza un ambiente di test personalizzato può contenere solo fino a 250 casi di test individuali. In caso contrario, l'esecuzione potrebbe essere saltata.

Limiti dei file

- Le dimensioni massime di un file di un'app che è possibile caricare è di 4 GB. Tieni presente che al momento non accettiamo file in formato.aab per Android.
- La dimensione massima del video generato automaticamente da Device Farm durante l'esecuzione del test è di 1 GB. Tutti i contenuti video rimanenti verranno troncati per qualsiasi video che superi questa dimensione. I clienti possono comunque utilizzare la propria soluzione di registrazione video, se presente, e archiviarla al di fuori dello storage gestito di Device Farm.
- La dimensione massima del registro del dispositivo generato automaticamente da Device Farm (logcat su Android o syslog su iOS) durante l'esecuzione del test è di 1 GB. Tutti i log che superano questa dimensione verranno troncati tutti i log rimanenti. Per i log di dimensioni superiori a 1 GB, i Clienti possono archiviare questi registri al di fuori dello storage gestito di Device Farm.
- La dimensione massima cumulativa degli artefatti dei clienti in modalità ambiente personalizzato di Device Farm è di 1 GB. Se questa dimensione viene superata dai tuoi artefatti, nessuno degli artefatti sarà disponibile.
- Se la dimensione cumulativa di tutti gli artefatti generati durante un'esecuzione di test supera i 4 GB, alcuni artefatti potrebbero essere eliminati (inclusi il video, i log dei dispositivi e gli artefatti dei clienti).

Limiti di API

- Device Farm segue un algoritmo token-bucket per limitare la frequenza delle chiamate API. Ad esempio, immagina di creare un bucket che contiene token. Ogni token rappresenta una transazione e una chiamata API utilizza un token. I token vengono aggiunti al bucket a una velocità fissa (ad esempio, 10 token al secondo) e il bucket ha una capacità massima (ad esempio, 100 token). Quando arriva una richiesta o un pacchetto, deve richiedere un token dal bucket per essere elaborato. Se i token sono sufficienti, la richiesta viene autorizzata e i token vengono rimossi. Se non ci sono abbastanza token, la richiesta viene ritardata o abbandonata, a seconda dell'implementazione.

  In Device Farm, ecco come viene implementato l'algoritmo:

  - Le richieste API Burst sono il numero massimo di richieste a cui il servizio è in grado di rispondere per un'API specificata in un ID di account cliente specificato. In altre parole, questa è la capacità del bucket. Puoi chiamare l'API tante volte quanti sono i token rimasti nel bucket e ogni richiesta consuma un token.

- La frequenza Transactions-per-second (TPS) è la velocità minima alla quale le richieste API possono essere eseguite. In altre parole, questa è la velocità con cui il bucket si riempie di token al secondo. Ad esempio, se un'API ha un numero burst pari a dieci ma un TPS pari a uno, puoi chiamarla dieci volte all'istante. Tuttavia, il bucket recupererebbe i token solo alla velocità di un token al secondo, con conseguente limitazione a una chiamata al secondo, a meno che tu non smetta di chiamare l'API per lasciare che il bucket si riempia.

Ecco le tariffe di Device Farm APIs:

- Per List and Get APIs, la capacità delle richieste dell'API Burst è `50` e la tariffa Transactions-per-second (TPS) è. `10`
- Per tutti gli altri APIs, la capacità delle richieste dell'API Burst è e `10` la velocità Transactions-per-second (TPS) è. `1`

Limiti degli endpoint Appium

I seguenti limiti si applicano a tutte le sessioni degli endpoint Appium. Per domande e indicazioni su come gestire al meglio i limiti, apri una richiesta di assistenza.

- Ogni comando Appium ha un limite di durata di esecuzione di 4 minuti, dopodiché il comando scade.
- L'endpoint accetta payload di input fino a 20 MB e consente payload di output fino a 20 MB. Qualsiasi richiesta con una dimensione di input o output maggiore di questa riceverà un errore di. WebDriver `'unsupported operation'`
- Le richieste vengono eseguite in sequenza sul dispositivo nell'ordine in cui vengono ricevute. Di conseguenza, consigliamo vivamente di inviare i comandi in sequenza e di attendere la risposta di ogni comando prima di inviarne uno nuovo. Detto questo, alcuni comandi del server Appium possono essere inviati in parallelo, in particolare:
  - getStatus
  - Ottieni sessioni
- L'endpoint non supporta il WebDriver BiDi protocollo in questo momento.
- L'endpoint non supporta plugin o driver Appium diversi dai driver and. XCUITest UIAutomator2
- È possibile utilizzare un massimo di 3 app come app ausiliarie con una richiesta di creazione di sessioni di accesso remoto. Detto questo, non c'è limite al numero di app che possono essere installate durante una sessione utilizzando l'`InstallToRemoteAccessSession`API.

Limiti delle variabili di ambiente personalizzate

I seguenti limiti si applicano a tutte le variabili di ambiente personalizzate. Per domande e indicazioni su come gestire al meglio i limiti, apri una richiesta di assistenza.

- È possibile configurare o eseguire un massimo di 32 variabili su un determinato progetto Device Farm.
- I nomi delle variabili non possono superare i 256 caratteri di lunghezza.
- I nomi delle variabili sono soggetti alle limitazioni imposte da`bash`. Vale a dire, devono contenere solo caratteri alfanumerici e caratteri di sottolineatura e non possono iniziare con un numero.
- I nomi delle variabili che iniziano con `$DEVICEFARM_` sono riservati all'uso interno del servizio.
- I valori delle variabili non possono superare i 256 caratteri di lunghezza.
- Le variabili di ambiente non possono essere utilizzate per configurare la selezione del calcolo dell'host di test nel file delle specifiche di test.

Strumenti e plugin per AWS Device Farm

Questa sezione contiene collegamenti e informazioni sull'utilizzo degli strumenti e dei plugin di AWS Device Farm. Puoi trovare i plugin Device Farm su AWS Labs su. GitHub

Se sei uno sviluppatore Android, abbiamo anche un'app di esempio AWS Device Farm per Android su GitHub. Puoi utilizzare l'app e i test di esempio come riferimento per i tuoi script di test di Device Farm.

Argomenti

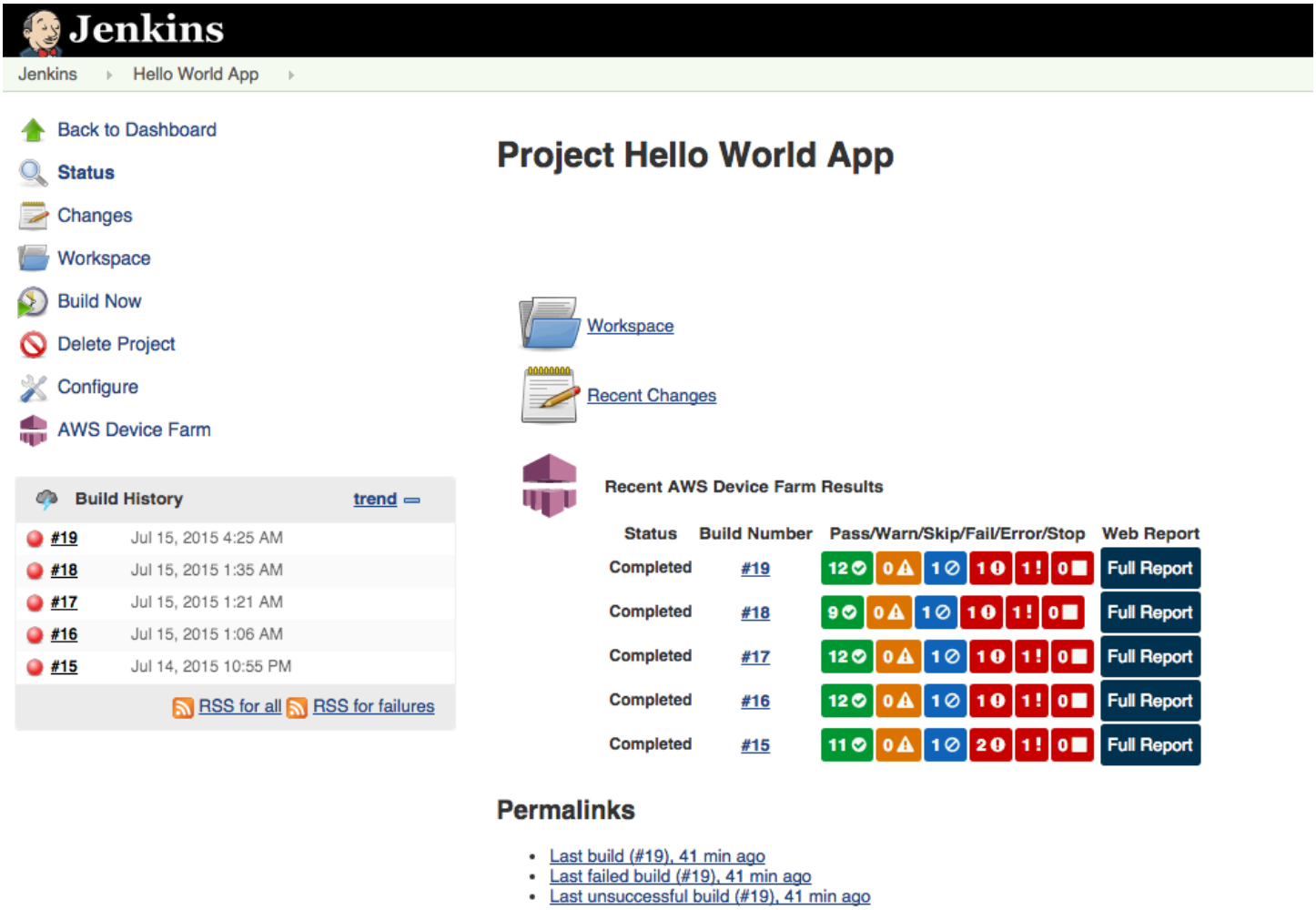
Integrazione di Device Farm con un server CI Jenkins

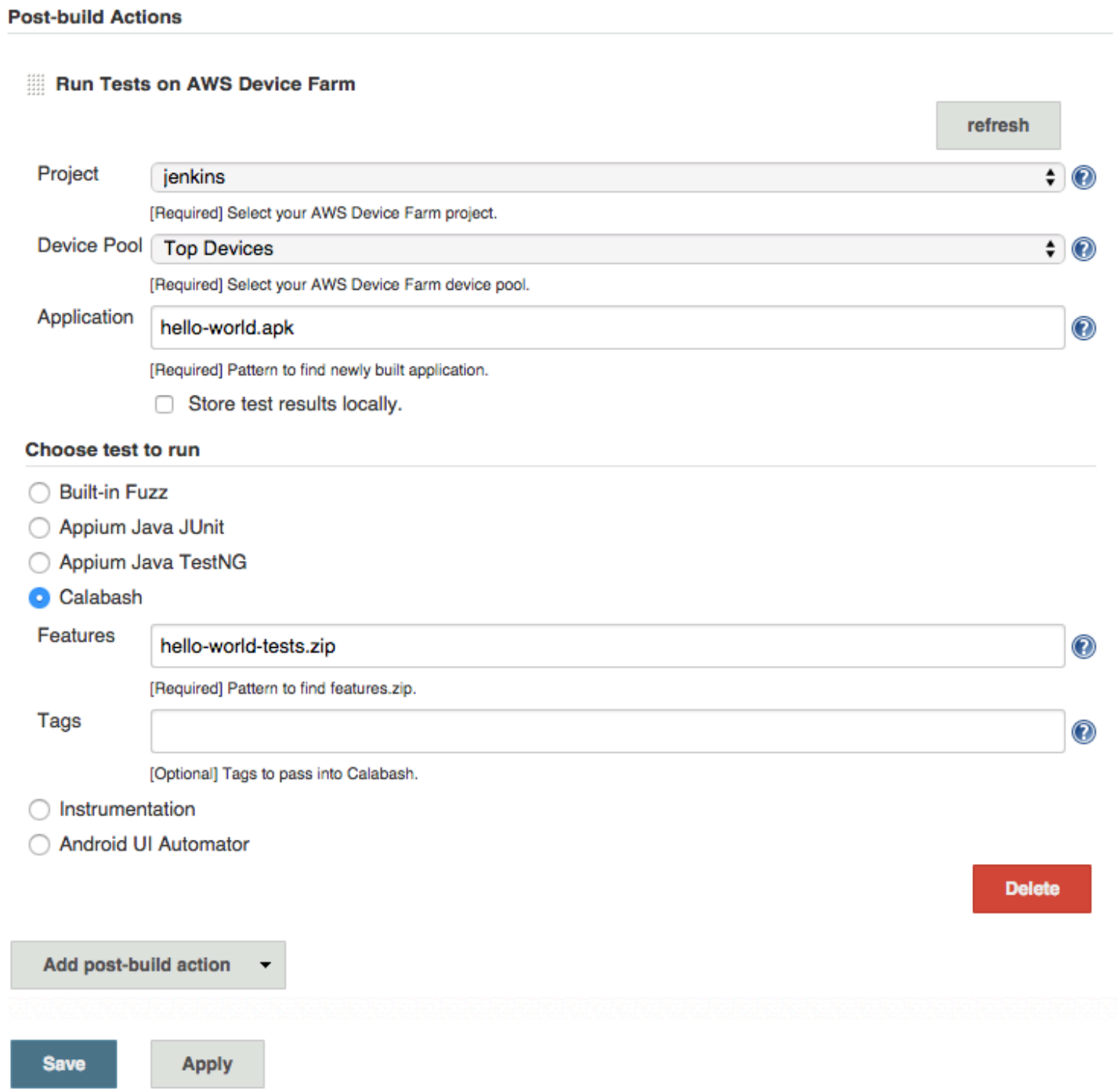
Il plug-in Jenkins CI fornisce la funzionalità AWS Device Farm dal tuo server di integrazione continua (CI) Jenkins. Per ulteriori informazioni, consulta Jenkins (software).

> **Note**
>
> Per scaricare il plugin Jenkins, vai a GitHube segui le istruzioni in. Fase 1: Installazione del plugin Jenkins CI per AWS Device Farm

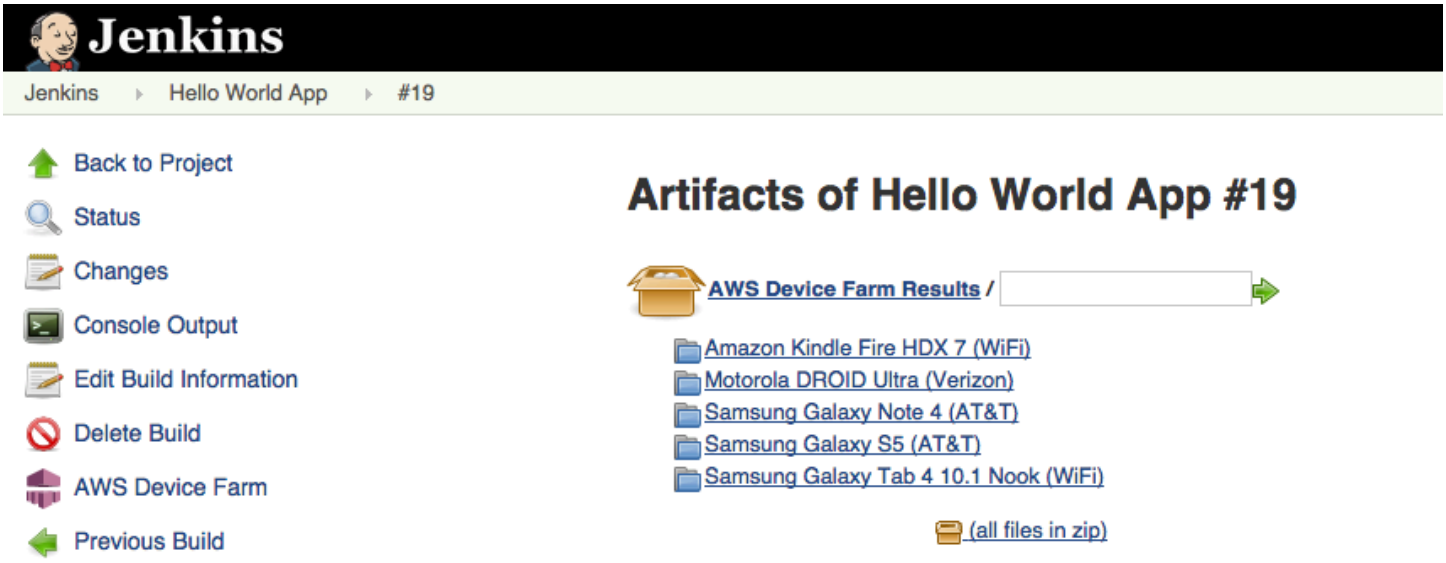
Questa sezione contiene una serie di procedure per configurare e utilizzare il plugin Jenkins CI con AWS Device Farm.

Le seguenti immagini mostrano le caratteristiche del plugin Jenkins CI.


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



Il plugin può anche aprire tutti gli artefatti del test (log, screenshot, ecc.) in locale:



Argomenti

Dipendenze

Il plugin Jenkins CI richiede AWS Mobile SDK 1.10.5 o versione successiva. Per ulteriori informazioni e per installare l'SDK, consulta SDK AWS Mobile.

Fase 1: Installazione del plugin Jenkins CI per AWS Device Farm

Esistono due opzioni per installare il plug-in Jenkins continuous integration (CI) per AWS Device Farm. Puoi cercare il plugin nella finestra di dialogo Available Plugins (Plugin disponibili) nell'interfaccia utente Web di Jenkins, oppure puoi scaricare il file `hpi` e installarlo in Jenkins.

Installazione dall'interfaccia utente di Jenkins

1. Trova il plugin nell'interfaccia utente di Jenkins selezionando Manage Jenkins (Gestisci Jenkins), Manage Plugins (Gestisci Plugin) e Available (Disponibili).

2. Cercare aws-device-farm.
3. Installa il plug-in AWS Device Farm.
4. Assicurati che il plugin sia di proprietà dell'utente `Jenkins`.
5. Riavvia Jenkins.

Scarica il plugin

1. Scaricate il `hpi` file direttamente da http://updates.jenkins-ci. org/latest/aws-device-farm.hpi.
2. Assicurati che il plugin sia di proprietà dell'utente `Jenkins`.
3. Installa il plugin utilizzando una delle seguenti opzioni.
   - Carica il plugin selezionando Manage Jenkins (Gestisci Jenkins), Manage Plugins (Gestisci plugin), Advanced (Avanzate) e Upload plugin (Carica plugin).
   - Inserisci il file `hpi` nella directory del plugin Jenkins (di solito `/var/lib/jenkins/plugins`).
4. Riavvia Jenkins.

Fase 2: creazione di un AWS Identity and Access Management utente per il plugin Jenkins CI per AWS Device Farm

Si consiglia di non utilizzare l'account AWS root per accedere a Device Farm. Invece, crea un nuovo utente AWS Identity and Access Management (IAM) (o utilizza un utente IAM esistente) nel tuo AWS account, quindi accedi a Device Farm con quell'utente IAM.

Per creare un nuovo utente IAM, consulta Creazione di un utente IAM (Console di gestione AWS). Assicurati di generare una chiave di accesso per ogni utente e di scaricare o salvare le credenziali di sicurezza degli utenti. Avrai bisogno delle credenziali in un secondo momento.

Autorizza l'utente IAM ad accedere a Device Farm

Per concedere all'utente IAM l'autorizzazione ad accedere a Device Farm, crea una nuova policy di accesso in IAM, quindi assegna la policy di accesso all'utente IAM come segue.

> **Note**
>
> L'account AWS root o l'utente IAM che utilizzi per completare i seguenti passaggi deve disporre dell'autorizzazione per creare la seguente policy IAM e collegarla all'utente IAM. Per ulteriori informazioni, consulta l'articolo relativo all'utilizzo delle policy.

Per creare la politica di accesso in IAM

1. Aprire la console IAM all'indirizzo https://console.aws.amazon.com/iam/.
2. Seleziona Policy.
3. Scegli Crea policy. (Se viene visualizzato il pulsante Get Started (Inizia), sceglierlo, quindi scegliere Create Policy (Crea policy)).
4. Accanto a Create Your Own Policy (Crea la tua policy) scegli Select (Seleziona).
5. In Policy Name (Nome policy) digitare un nome per la policy, ad esempio **`AWSDeviceFarmAccessPolicy`**.
6. Per Descrizione, digita una descrizione che ti aiuti ad associare questo utente IAM al tuo progetto Jenkins.
7. Per Policy Document (Documento della policy), digita la seguente istruzione:

   JSON

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

8. Scegli Crea policy.

Per assegnare la politica di accesso all'utente IAM

1. Aprire la console IAM all'indirizzo https://console.aws.amazon.com/iam/.
2. Scegliere Users (Utenti).
3. Scegli l'utente IAM a cui assegnare la politica di accesso.
4. Nell'area Permissions (Autorizzazioni), in Managed Policies (Policy gestite), seleziona Attach Policy (Collega policy).
5. Seleziona la policy che hai appena creato (ad esempio, AWSDeviceFarmAccessPolicy).
6. Scegli Attach Policy (Collega policy).

Fase 3: Configurazione del plugin Jenkins CI per la prima volta in AWS Device Farm

La prima volta che esegui il server Jenkins, dovrai configurare il sistema nel seguente modo.

> **Note**
>
> Se stai utilizzando slot per i dispositivi, la loro funzionalità sarà disabilitata per impostazione predefinita.

1. Accedi all'interfaccia utente Web di Jenkins.
2. Sul lato sinistro dello schermo, scegli Manage Jenkins (Gestisci Jenkins).
3. Scegli Configure System (Configura sistema).
4. Scorri verso il basso fino all'intestazione di AWS Device Farm.
5. Copia le tue credenziali di sicurezza da Creazione di un utente IAM per il tuo plugin Jenkins CI e incolla il tuo ID chiave di accesso e la tua chiave di accesso segreta nelle rispettive caselle.
6. Scegli Save (Salva).

Fase 4: Usare il plugin in un job Jenkins

Dopo aver installato il plugin Jenkins, segui queste istruzioni per utilizzare il plugin in un'attività di Jenkins.

1. Accedi all'interfaccia utente Web di Jenkins.
2. Clicca sull'attività che desideri modificare.
3. Sul lato sinistro dello schermo, scegli Configure (Configura).
4. Scorri verso il basso fino all'intestazione Post-build Actions (Operazioni post-compilazione).
5. Fai clic su Aggiungi azione post-compilazione e seleziona Run Tests on AWS Device Farm.
6. Seleziona il progetto che desideri utilizzare.
7. Seleziona il pool di dispositivi che desideri utilizzare.
8. Seleziona se archiviare in locale gli artefatti dei test (ad esempio, i log e gli screenshot).
9. In Application (Applicazione), inserisci il percorso dell'applicazione compilata.
10. Seleziona il test che desideri eseguire e compila tutti i campi obbligatori.
11. Scegli Save (Salva).

Integrazione di Device Farm con un sistema di build Gradle

Il plugin Device Farm Gradle fornisce l'integrazione di AWS Device Farm con il sistema di build Gradle in Android Studio. Per ulteriori informazioni, consulta Gradle.

> **Note**
>
> Per scaricare il plugin Gradle, vai a GitHube segui le istruzioni in. Creazione del plugin Device Farm Gradle

Il plugin Device Farm Gradle fornisce la funzionalità Device Farm dal tuo ambiente Android Studio. Puoi iniziare i test su telefoni e tablet Android reali ospitati da Device Farm.

Questa sezione contiene una serie di procedure per configurare e utilizzare il Device Farm Gradle Plugin.

Argomenti

- Dipendenze
- Fase 1: creazione del plug-in AWS Device Farm Gradle
- Fase 2: configurazione del plugin AWS Device Farm Gradle

- Fase 3: Generazione di un utente IAM nel plugin Device Farm Gradle
- Fase 4: Configurazione dei tipi di test

Dipendenze

Runtime

- Il Device Farm Gradle Plugin richiede AWS Mobile SDK 1.10.15 o successivo. Per ulteriori informazioni e per installare l'SDK, consulta SDK AWS Mobile.
- Android Tools Builder Test API 0.5.2
- Apache Commons Lang3 3.3.4

Per unit test

- Testng 6.8.8
- Jmockit 1.19
- Android Gradle Tools 1.3.0

Fase 1: creazione del plug-in AWS Device Farm Gradle

Questo plugin fornisce l'integrazione di AWS Device Farm con il sistema di build Gradle in Android Studio. Per ulteriori informazioni, consulta Gradle.

> **Note**
>
> La creazione del plugin è facoltativa. Il plugin viene pubblicato tramite Maven Central. Se desideri consentire a Gradle di scaricare direttamente il plugin, salta questa fase e passa a Fase 2: configurazione del plugin AWS Device Farm Gradle.

Per creare il plugin

1. Vai al GitHubrepository e clona.
2. Creare il plugin usando `gradle install`.

   Il plugin viene installato sull'archivio Maven locale.

Fase successiva: Fase 2: configurazione del plugin AWS Device Farm Gradle

Fase 2: configurazione del plugin AWS Device Farm Gradle

Se non lo hai già fatto, clona l'archivio e installa il plugin usando la procedura indicata qui: Creazione del plugin Device Farm Gradle.

Per configurare il plugin AWS Device Farm Gradle

1. Aggiungere l'artefatto del plugin al proprio elenco di dipendenze in `build.gradle`.

```
    buildscript {

        repositories {
            mavenLocal()
            mavenCentral()
        }

        dependencies {
            classpath 'com.android.tools.build:gradle:1.3.0'
            classpath 'com.amazonaws:aws-devicefarm-gradle-plugin:1.0'
        }
    }
```

2. Configurare il plugin nel proprio file `build.gradle`. La seguente configurazione della specifica di test deve fungere da guida:

```
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. Esegui il test di Device Farm utilizzando la seguente attività:`gradle devicefarmUpload`.

   L'output della build stamperà un collegamento alla console Device Farm dove è possibile monitorare l'esecuzione del test.

Fase successiva: Generazione di un utente IAM nel plugin Device Farm Gradle

Fase 3: Generazione di un utente IAM nel plugin Device Farm Gradle

AWS Identity and Access Management (IAM) ti aiuta a gestire le autorizzazioni e le politiche per lavorare con AWS le risorse. Questo argomento illustra come generare un utente IAM con autorizzazioni per accedere alle risorse di AWS Device Farm.

Se non l'hai già fatto, completa i passaggi 1 e 2 prima di generare un utente IAM.

Si consiglia di non utilizzare l'account AWS root per accedere a Device Farm. Invece, crea un nuovo utente IAM (o utilizza un utente IAM esistente) nel tuo AWS account, quindi accedi a Device Farm con quell'utente IAM.

> **Note**
>
> L'account AWS root o l'utente IAM che utilizzi per completare i seguenti passaggi deve disporre dell'autorizzazione per creare la seguente policy IAM e collegarla all'utente IAM. Per ulteriori informazioni, consulta l'articolo relativo all'utilizzo delle policy.

Per creare un nuovo utente con la politica di accesso appropriata in IAM

1. Aprire la console IAM all'indirizzo https://console.aws.amazon.com/iam/.
2. Scegliere Users (Utenti).
3. Scegliere Create New Users (Crea nuovi utenti).
4. Immettere il nome utente di propria scelta.

   Ad esempio, **`GradleUser`**.
5. Scegli Create (Crea).
6. Scegliere Download Credentials (Scarica credenziali) e salvarle in una posizione in cui saranno facilmente recuperabili in seguito.
7. Scegli Chiudi.
8. Selezionare il nome utente nell'elenco.
9. In Permissions (Autorizzazioni), espandere l'intestazione Inline Policies (Policy inline) facendo clic sulla freccia giù a destra.
10. Scegli Fai clic qui dove dice, Non ci sono politiche in linea da mostrare. Per crearne uno, clicca qui.
11. Sulla schermata Set Permissions (Imposta autorizzazioni), selezionare Custom Policy (Personalizza policy).
12. Scegli Seleziona.
13. Assegnare un nome alla policy, ad esempio **`AWSDeviceFarmGradlePolicy`**.
14. Incollare la seguente policy in Policy Document (Documento policy).

    JSON

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

15. Scegli Apply Policy (Applica policy).

Fase successiva: Configurazione dei tipi di test.

Per ulteriori informazioni, consulta Creazione di un utente IAM (Console di gestione AWS) oConfigurazione.

Fase 4: Configurazione dei tipi di test

Per impostazione predefinita, il plug-in AWS Device Farm Gradle esegue il Strumentazione per Android e AWS Device Farm test. Se desideri eseguire i tuoi test o specificare parametri aggiuntivi, puoi scegliere di configurare un tipo di test. Questo argomento fornisce informazioni su ciascun tipo di test disponibile e su ciò che devi fare in Android Studio per configurarlo per l'uso. Per ulteriori informazioni sui tipi di test disponibili in Device Farm, vedereFramework di test e test integrati in AWS Device Farm.

Se non l'hai già fatto, completa i passaggi da 1 a 3 prima di configurare i tipi di test.

> **Note**
>
> Se stai utilizzando slot per i dispositivi, la loro funzionalità sarà disabilitata per impostazione predefinita.

Appium

Device Farm fornisce supporto per Appium Java e JUnit TestNG per Android.

- Appium (in Java ()) JUnit
- Appium (in Java (TestNg))

Puoi scegliere `useTestNG()` o `useJUnit()`. `JUnit` è l'impostazione predefinita e non deve essere specificata in modo esplicito.

```
    appium {
        tests file("path to zip file") // required
        useTestNG() // or useJUnit()
    }
```

Incorporato: fuzz

Device Farm offre un tipo di fuzz test integrato, che invia in modo casuale gli eventi dell'interfaccia utente ai dispositivi e quindi riporta i risultati.

```
    fuzz {

       eventThrottle 50 // optional default
       eventCount 6000  // optional default
       randomizerSeed 1234 // optional default blank

     }
```

Per ulteriori informazioni, consulta Esecuzione del fuzz test integrato di Device Farm (Android e iOS).

Instrumentation

Device Farm fornisce supporto per la strumentazione (EspressoJUnit, Robotium o qualsiasi test basato sulla strumentazione) per Android. Per ulteriori informazioni, consulta Strumentazione per Android e AWS Device Farm.

Quando si esegue un test di strumentazione in Gradle, Device Farm utilizza il `.apk` file generato dalla directory AndroidTest come fonte dei test.

```
    instrumentation {

        filter "test filter per developer docs" // optional

    }
```

Cronologia dei documenti di AWS Device Farm

La tabella seguente descrive le modifiche importanti apportate alla documentazione dall'ultima versione della guida.

Modifica	Descrizione	Data della modifica
supporto per endpoint Appium	Device Farm offre ora un endpoint Appium completamente gestito per il test remoto dei dispositivi, che consente lo sviluppo e il debug rapidi dei test. Ciò integra il metodo di esecuzione lato server esistente, in cui i test vengono caricati ed eseguiti direttamente su Device Farm. Sebbene l'esecuzione lato server sia ideale per CI/CD pipeline e test su larga scala, il nuovo endpoint Appium locale consente l'iterazione e lo sviluppo più rapidi dei test su dispositivi reali.	17 novembre 2025
Miglioramenti all'host di test iOS	Device Farm ora supporta un'esperienza aggiornata per l'ambiente di test iOS, che consente la coerenza nelle configurazioni tra i test Android e iOS. Per ulteriori informazioni, consulta Host per ambienti di test personalizzati. Inoltre, le informazioni relative agli host di test Android ritirati sono state rimosse. Gli utenti Android sono incoraggiati a utilizzare gli host di test di Amazon Linux 2.	31 ottobre 2025
AL2 supporto	Device Farm ora supporta l'ambiente di AL2 test per Android. Ulteriori informazioni su AL2.	6 novembre 2023
Migrazione da ambienti di test standard a ambienti di test personalizzati	Guida alla migrazione aggiornata all'obsolescenza dei documenti per i test in modalità standard nel dicembre 2023.	3 settembre 2023
Supporto VPC ENI	Device Farm ora consente ai dispositivi privati di utilizzare la funzionalità di connettività VPC-ENI per aiutare i clienti a	15 maggio 2023

Modifica	Descrizione	Data della modifica
	connettersi in modo sicuro ai propri endpoint privati ospitati su AWS, software on-premise o un altro provider di cloud. Scopri di più su VPC-ENI.	
Aggiornamenti Polaris UI	La console Device Farm ora supporta il framework Polaris.	28 luglio 2021
Supporto Python 3	Device Farm ora supporta Python 3 nei test in modalità personalizzata. Ulteriori informazioni sull'utilizzo di Python 3 nei pacchetti di test: • Appium (Python) • Appium (Python)	20 aprile 2020
Nuove informazioni sulla sicurezza e informazioni sull'etic hettatura delle risorse AWS .	Per rendere la protezione dei AWS servizi più semplice e completa, è stata creata una nuova sezione sulla sicurezza . Per ulteriori informazioni, consulta Sicurezza in AWS Device Farm È stata aggiunta una nuova sezione sull'etichettatura in Device Farm. Per ulteriori informazioni sul tagging, consulta Inserimento di tag in Device Farm.	27 marzo 2020
Rimozione di Direct Device Access.	L'accesso diretto ai dispositivi (debug remoto su dispositi vi privati) non è più disponibile per l'utilizzo generale. Per ulteriori informazioni sulla disponibilità futura dell'accesso diretto ai dispositivi, contattaci.	9 settembre 2019
Aggiornamento della configurazione del plugin Gradle	Una configurazione del plugin Gradle modificata ora include una versione personalizzabile della configura zione Gradle, con parametri opzionali commentati. Ulteriori informazioni su Configurazione del plugin Device Farm Gradle.	16 agosto 2019

Modifica	Descrizione	Data della modifica
Nuovo requisito per i test eseguiti con XCTest	Per le esecuzioni di test che utilizzano il XCTest framework , Device Farm ora richiede un pacchetto di app creato per i test. Ulteriori informazioni su the section called “XCTest”.	4 febbraio 2019
Supporto per i tipi di test Appium Node.js e Appium Ruby in ambienti personalizzati	È ora possibile eseguire i test in entrambi gli ambienti di test personalizzati Appium Node.js e Appium Ruby. Ulteriori informazioni su Framework di test e test integrati in AWS Device Farm.	10 gennaio 2019
Supporto per server Appium versione 1.7.2 sia in ambienti standard che personalizzati. Supporto per versione 1.8.1 utilizzando un file YAML di specifica di test personalizzato in un ambiente di test personalizzato.	Ora puoi eseguire i test sia in ambienti standard che personalizzati con server Appium versioni 1.72, 1.71 e 1.6.5. È inoltre possibile eseguire i test con versioni 1.8.1 e 1.8.0 utilizzando un file YAML di specifica di test personalizzato in un ambiente personalizzato. Ulteriori informazioni su Framework di test e test integrati in AWS Device Farm.	2 ottobre 2018
Ambienti di test personalizzati	Con un ambiente di test personalizzato, puoi assicurarti che i test vengano eseguiti come nel tuo ambiente locale. Device Farm ora fornisce supporto per i log live e lo streaming video, in modo da poter ottenere un feedback immediato sui test eseguiti in un ambiente di test personalizzato. Ulteriori informazioni su Ambienti di test personalizzati in AWS Device Farm.	16 agosto 2018

Modifica	Descrizione	Data della modifica
Support per l'utilizzo di Device Farm come fornitore AWS CodePipeline di test	Ora puoi configurare una pipeline AWS CodePipeline per utilizzare le esecuzioni di AWS Device Farm come azioni di test nel processo di rilascio. CodePipeline ti consente di collegare rapidamente il tuo repository alle fasi di compilazione e test per ottenere un sistema di integrazione continua personalizzato in base alle tue esigenze. Ulteriori informazioni su Integrazione di AWS Device Farm in una fase di CodePipeline test.	19 luglio 2018
Supporto per dispositivi privati	È ora possibile utilizzare i dispositivi privati per pianificare le sessioni di test e avviare le sessioni di accesso remoto. Puoi gestire profili e impostazioni per questi dispositivi, creare endpoint Amazon VPC per testare app private e creare sessioni di debug remote. Ulteriori informazioni su Dispositivi privati in AWS Device Farm.	02 maggio 2018
Supporto per Appium 1.6.3	È ora possibile impostare la versione di Appium per i tuoi test Appium personalizzati.	21 marzo 2017
Imposta il timeout di esecuzione per le sessioni di test	È possibile impostare il timeout di esecuzione di una sessione di test o per tutti i test in un progetto. Ulteriori informazioni su Impostazione del timeout di esecuzione per le esecuzioni di test in AWS Device Farm.	9 febbraio 2017
Configurazione di reti	È ora possibile simulare le condizioni e le connessioni di rete per una sessione di test. Ulteriori informazioni su Simulazione delle connessioni e delle condizioni di rete per le esecuzioni di AWS Device Farm.	8 dicembre 2016
Nuova sezione di risoluzione dei problemi	Ora puoi risolvere i problemi di caricamento dei pacchetti di test utilizzando una serie di procedure progettate per risolvere i messaggi di errore che potresti riscontrare nella console Device Farm. Ulteriori informazioni su Risoluzione degli errori di Device Farm.	10 agosto 2016

Modifica	Descrizione	Data della modifica
Sessioni di accesso remoto	È ora possibile accedere da remoto e interagire con un singolo dispositivo nella console. Ulteriori informazioni su Accesso remoto.	19 aprile 2016
Slot di dispositivi self-service	Ora puoi acquistare slot per dispositivi utilizzando Console di gestione AWS, the o l' AWS Command Line Interface API. Ulteriori informazioni su come Acquisto di uno slot per dispositivi in Device Farm.	22 marzo 2016
Come arrestare sessioni di test	Ora puoi interrompere le esecuzioni dei test utilizzando l' Console di gestione AWS AWS Command Line Interface , la o l'API. Ulteriori informazioni su come Interruzione di un'esecuzione in AWS Device Farm.	22 marzo 2016
XCTest Nuovi tipi di test dell'interfaccia utente	Ora puoi eseguire test personalizzati XCTest dell'inte rfaccia utente sulle applicazioni iOS. Ulteriori informazioni sul tipo di test Integrazione dell' XCTest interfaccia utente per iOS con Device Farm.	8 marzo 2016
Nuovi tipi di test per Appium Python	È ora possibile eseguire test personalizzati per Appium Python su Android, iOS e applicazioni Web. Ulteriori informazioni su Framework di test e test integrati in AWS Device Farm.	19 gennaio 2016
Tipi di test per applicazioni Web	Ora puoi eseguire test personalizzati Appium Java e JUnit TestNg su applicazioni web. Ulteriori informazioni su Test di app Web in AWS Device Farm.	19 Novembre 2015
Plugin AWS Device Farm Gradle	Ulteriori informazioni su come installare e utilizzare il Plugin Device Farm Gradle.	28 settembre 2015
Nuovo test integrato per Android: Explorer	Il test Explorer ricerca per indicizzazione nell'app analizzan do ogni schermata come se fosse un utente finale e acquisendo screenshot durante l'analisi.	16 settembre 2015

Modifica	Descrizione	Data della modifica
Supporto iOS aggiunto	Scopri di più sul test dei dispositivi iOS e sull'esecuzione di test iOS (inclusi XCTest) inFramework di test e test integrati in AWS Device Farm.	4 agosto 2015
Versione pubblica iniziale	Questa è la versione pubblica iniziale della AWS Device Farm Developer Guide.	13 luglio 2015

AWS Glossario

Per la AWS terminologia più recente, consultate il AWS glossario nella sezione Reference.Glossario AWS

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.