



Guida per gli sviluppatori

AWS Device Farm



Versione API 2015-06-23

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

AWS Device Farm: Guida per gli sviluppatori

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

I marchi e il trade dress di Amazon non possono essere utilizzati in relazione ad alcun prodotto o servizio che non sia di Amazon, in alcun modo che possa causare confusione tra i clienti, né in alcun modo che possa denigrare o screditare Amazon. Tutti gli altri marchi non di proprietà di Amazon sono di proprietà delle rispettive aziende, che possono o meno essere associate, collegate o sponsorizzate da Amazon.

Table of Contents

Cos'è AWS Device Farm?	1
Test automatizzato delle app	1
Interazione con accesso remoto	1
Terminologia	2
Configurazione	3
Configurazione	4
Passaggio 1: iscriviti a AWS	4
Passaggio 2: crea o utilizza un IAM utente nel tuo AWS account	4
Passaggio 3: autorizzare l'IAMutente ad accedere a Device Farm	5
Approfondimenti	6
Nozioni di base	7
Prerequisiti	7
Fase 1: Accesso alla console di	8
Fase 2: Creare un progetto	8
Fase 3: Creare e iniziare una corsa	8
Passaggio 4: Visualizza i risultati della corsa	10
Fasi successive	10
Acquista slot per dispositivi	11
Acquista slot per dispositivi (console)	11
Acquista uno slot per dispositivo (AWS CLI)	13
Acquista uno slot per dispositivo (API)	17
Annullare lo slot di un dispositivo (console)	18
Annulla lo slot di un dispositivo (AWS CLI)	18
Annullare lo slot di un dispositivo () API	18
Concetti	19
Dispositivi	19
Dispositivi supportati	19
Pool di dispositivi	20
Dispositivi privati	20
Branding del dispositivo	20
Slot per dispositivi	20
App preinstallate per dispositivi	21
Funzionalità del dispositivo	21
Ambienti di test	21

Ambiente di test standard	21
Ambiente di test personalizzato	22
Esecuzioni	22
Esegui la configurazione	23
Esegui la conservazione dei file	23
Esegui lo stato del dispositivo	23
Esecuzioni parallele	23
Impostazione del timeout di esecuzione	23
Annunci nelle puntate	23
File multimediali in tirature	24
Attività comuni per le esecuzioni	24
App	24
App di strumentazione	24
Riassegnare la firma delle app durante le esecuzioni	24
App offuscate in corso di esecuzione	25
Report	25
Conservazione dei report	25
Componenti dei report	25
Registra i report di accesso	25
Attività comuni per i report	25
Sessioni	26
Dispositivi supportati per l'accesso remoto	26
Conservazione dei file di sessione	26
App di strumentazione	26
Riassegnare la firma delle app nelle sessioni	27
App offuscate nelle sessioni	27
Lavorare con progetti	28
Crea un progetto	28
Prerequisiti	28
Crea un progetto (console)	28
Crea un progetto (AWS CLI)	29
Crea un progetto (API)	29
Visualizza l'elenco dei progetti	29
Prerequisiti	30
Visualizza l'elenco dei progetti (console)	30
Visualizza l'elenco dei progetti (AWS CLI)	30

Visualizza l'elenco dei progetti (API)	30
Utilizzo delle esecuzioni di test	31
Crea un'esecuzione di test	31
Prerequisiti	32
Crea un'esecuzione di test (console)	32
Crea un test run (AWS CLI)	35
Creare un test run (API)	45
Passaggi successivi	46
Imposta il timeout di esecuzione	46
Prerequisiti	47
Imposta il timeout di esecuzione per un progetto	47
Imposta il timeout di esecuzione per un'esecuzione di test	47
Simula le connessioni e le condizioni di rete	48
Imposta la configurazione della rete quando pianifichi un'esecuzione di test	48
Crea un profilo di rete	49
Modifica le condizioni della rete durante il test	51
Interrompi una corsa	51
Interrompi una corsa (console)	51
Interrompi una corsa (AWS CLI)	53
Interrompere una corsa (API)	55
Visualizza un elenco di esecuzioni	55
Visualizza un elenco di esecuzioni (console)	55
Visualizza un elenco di esecuzioni (AWS CLI)	55
Visualizza un elenco di esecuzioni (API)	56
Creare un pool di dispositivi	56
Prerequisiti	56
Crea un pool di dispositivi (console)	56
Crea un pool di dispositivi (AWS CLI)	58
Creare un pool di dispositivi (API)	59
Analisi dei risultati	59
Utilizzo dei rapporti di prova	59
Lavorare con gli artefatti	69
Etichettatura in Device Farm	74
Assegnazione di tag alle risorse	74
Ricerca di risorse per tag	75
Rimozione dei tag dalle risorse	76

Tipi e framework di test	77
Framework di test	77
Framework di test delle applicazioni Android	77
Framework di test delle applicazioni iOS	77
Framework di test delle applicazioni Web	77
Framework in un ambiente di test personalizzato	77
Supporto per la versione Appium	77
Tipi di test integrati	78
Appium	78
Supporto versione	78
Configura il tuo pacchetto di test Appium	79
Crea un file di pacchetto compresso	89
Carica il tuo pacchetto di test su Device Farm	92
Acquisisci schermate dei tuoi test (opzionale)	94
Test Android	94
Framework di test delle applicazioni Android	94
Tipi di test integrati per Android	94
Instrumentation	94
Test iOS	97
Framework di test delle applicazioni iOS	97
Tipi di test integrati per iOS	98
XCTest	98
XCTestInterfaccia utente	100
Test delle app Web	102
Regole per i dispositivi misurati e non misurati	102
Test integrati	103
Tipi di test integrati	103
Integrato: fuzz (Android e iOS)	103
Lavorare con ambienti di test personalizzati	105
Sintassi delle specifiche di test	106
Esempio di specifiche di test	108
Ambiente di test Android	113
Lingue e versioni di runtime supportate	114
Intervalli IP supportati	116
devicefarm-cli	116
Selezione dell'host di test Android	118

Esempio di file di specifiche di test	119
Migrazione ad Amazon Linux 2 Test Host	123
Variabili di ambiente	126
Variabili di ambiente comuni	126
Variabili di ambiente Appium Java JUnit	128
Variabili di ambiente Appium Java TestNg	128
Variabili di ambiente XCUITest	128
Migrazione dei test	129
Considerazioni sulla migrazione	129
Fasi della migrazione	131
Struttura Appium	131
Strumentazione Android	131
Migrazione dei test XCUITest iOS esistenti	132
Estensione della modalità personalizzata	132
Impostazione di un PIN	132
Accelerazione dei test basati su Appium grazie alle funzionalità desiderate	133
Utilizzo di Webhook e altro dopo l'esecuzione dei test APIs	135
Aggiungere file aggiuntivi al pacchetto di test	136
Lavorare con l'accesso remoto	140
Crea una sessione	140
Prerequisiti	141
Crea una sessione con la console Device Farm	141
Fasi successive	141
Usa una sessione	142
Prerequisiti	142
Usa una sessione nella console Device Farm	142
Fasi successive	143
Suggerimenti e trucchi	143
Ottieni i risultati della sessione	143
Prerequisiti	144
Visualizzazione dei dettagli della sessione	144
Scaricamento del video o dei registri della sessione	144
Lavorare con dispositivi privati	145
Gestione dei dispositivi privati	146
Creazione di un profilo dell'istanza	146
Gestire un'istanza di dispositivo privato	148

Creare un'esecuzione di test o una sessione di accesso remoto	150
Passaggi successivi	151
Selezione di dispositivi privati	151
ARNRegole del dispositivo	152
Regole delle etichette delle istanze del dispositivo	152
ARNRegole dell'istanza	153
Crea un pool di dispositivi privato	154
Creazione di un pool di dispositivi privati con dispositivi privati (AWS CLI)	156
Creazione di un pool di dispositivi privato con dispositivi privati (API)	156
Ignorare la rifirma dell'app	156
Salta la rifirma dell'app sui dispositivi Android	159
Ignora la nuova firma dell'app sui dispositivi iOS	159
Crea una sessione di accesso remoto per rendere affidabile la tua app	160
Lavorare in più regioni	161
VPCpanoramica del peering	161
Prerequisiti	163
Fase 1: Stabilire una connessione peering tra due VPCs	163
Fase 2: Aggiornare le tabelle delle rotte per VPC -1 e VPC -2	164
Fase 3: Creazione di gruppi target	164
Fase 4: Creare un Network Load Balancer	166
Fase 5: Creare un servizio VPC endpoint	167
Fase 6: Creare una configurazione dell'VPCendpoint nell'applicazione	167
Fase 7: Creare un'esecuzione di test	168
Creazione di sistemi scalabili VPC	168
Disattivazione dei dispositivi privati	168
VPCconnettività	170
AWS controllo degli accessi e IAM	172
Ruoli collegati ai servizi	173
Autorizzazioni di ruolo collegate ai servizi per Device Farm	174
Creazione di un ruolo collegato al servizio per Device Farm	177
Modifica di un ruolo collegato al servizio per Device Farm	177
Eliminazione di un ruolo collegato al servizio per Device Farm	177
Regioni supportate per i ruoli collegati al servizio Device Farm	178
Prerequisiti	179
Connessione ad Amazon VPC	180
Limiti	181

Utilizzo dei servizi VPC endpoint - Legacy	182
Prima di iniziare	183
Fase 1: Creazione di un Network Load Balancer	184
Fase 2: Creare un servizio VPC endpoint	186
Fase 3: Creare una configurazione dell'VPCendpoint	187
Fase 4: Creare un'esecuzione di test	189
Registrazione delle chiamate API con AWS CloudTrail	190
Informazioni su AWS Device Farm inCloudTrail	190
Comprendere le voci dei file di log di AWS Device Farm	191
Integrazione di CodePipeline	194
ConfigurareCodePipelineper utilizzare i test di Device Farm	195
Riferimento AWS CLI	199
finestrePowerShellriferimento	200
Automazione di Device Farm	201
Esempio: utilizzo diAWSSDK per avviare una Device Farm, eseguire e raccogliere artefatti	201
Risoluzione dei problemi	206
Applicazioni Android	206
ANDROID_APP_UNZIP_FAILED	206
ANDROID_APP_AAPT_DEBUG_BADGING_FAILED	207
ANDROID_APP_PACKAGE_NAME_VALUE_MISSING	208
ANDROID_APP_SDK_VERSION_VALUE_MISSING	209
ANDROID_APP_AAPT_DUMP_XMLTREE_FAILED	210
ANDROID_APP_DEVICE_ADMIN_PERMISSIONS	211
Alcune finestre della mia applicazione Android mostrano una schermata vuota o nera	213
Appium Java JUnit	213
APPIUM_JAVA_JUNIT_TEST_PACKAGE_PACKAGE_UNZIP_FAILED	213
APPIUM_JAVA_JUNIT_TEST_PACKAGE_DEPENDENCY_DIR_MISSING	214
APPIUM_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_DEPENDENCY_IN_DIR	215
APPIUM_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING	216
APPIUM_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_TESTS_IN_JAR	218
APPIUM_JAVA_JUNIT_TEST_PACKAGE_JUNIT_VERSION_VALUE_UNKNOWN	219
APPIUM_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION	220
Appium Java web JUnit	222
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED	222

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_DEPENDENCY_DIR_MISSING	223
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_DEPENDENCY_IN_	
_DIR	224
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING	225
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_TESTS	
_IN_JAR	226
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_JUNIT_VERSION_VALUE_UNKNOWN ...	227
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION	228
Appium Java TestNG	229
APPIUM_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED	230
APPIUM_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSING	231
APPIUM_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_DEPENDENCY_IN_	
DIR	232
APPIUM_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING	233
APPIUM_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_TESTS_IN_	
JAR	234
Appium Java TestNg web	235
APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED	236
APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSING	237
APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_DEPENDENCY	
_IN_DIR	238
APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING	239
APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_TESTS	
_IN_JAR	240
Appium Python	241
APPIUM_PYTHON_TEST_PACKAGE_UNZIP_FAILED	241
APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING	242
APPIUM_PYTHON_TEST_PACKAGE_INVALID_PLATFORM	243
APPIUM_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING	245
APPIUM_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME	246
APPIUM_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISSING	247
APPIUM_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION	248
APPIUM_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS_FAILED	249
APPIUM_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED	250
APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEELS_INSUFFICIENT	252
Web Appium Python	253

APPIUM_WEB_PYTHON_TEST_PACKAGE_UNZIP_FAILED	253
APPIUM_WEB_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING	254
APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PLATFORM	255
APPIUM_WEB_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING	256
APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME	257
APPIUM_WEB_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISSING	258
APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION	259
APPIUM_WEB_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS_FAILED	260
APPIUM_WEB_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED	261
Instrumentation	263
INSTRUMENTATION_TEST_PACKAGE_UNZIP_FAILED	263
INSTRUMENTATION_TEST_PACKAGE_AAPT_DEBUG_BADGING_FAILED	264
INSTRUMENTATION_TEST_PACKAGE_INSTRUMENTATION_RUNNER_VALUE_MISSING	265
INSTRUMENTATION_TEST_PACKAGE_AAPT_DUMP_XMLTREE_FAILED	266
INSTRUMENTATION_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSING	267
Applicazioni iOS	268
IOS_APP_UNZIP_FAILED	269
IOS_APP_PAYLOAD_DIR_MISSING	269
IOS_APP_APP_DIR_MISSING	270
IOS_APP_PLIST_FILE_MISSING	271
IOS_APP_CPU_ARCHITECTURE_VALUE_MISSING	272
IOS_APP_PLATFORM_VALUE_MISSING	273
IOS_APP_WRONG_PLATFORM_DEVICE_VALUE	275
IOS_APP_FORM_FACTOR_VALUE_MISSING	276
IOS_APP_PACKAGE_NAME_VALUE_MISSING	277
IOS_APP_EXECUTABLE_VALUE_MISSING	279
XCTest	280
XCTEST_TEST_PACKAGE_UNZIP_FAILED	280
XCTEST_TEST_PACKAGE_XCTEST_DIR_MISSING	281
XCTEST_TEST_PACKAGE_PLIST_FILE_MISSING	282
XCTEST_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING	283
XCTEST_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING	284
XCTest UI	285
XCTEST_UI_TEST_PACKAGE_UNZIP_FAILED	286
XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_MISSING	286
XCTEST_UI_TEST_PACKAGE_APP_DIR_MISSING	287

XCTEST_UI_TEST_PACKAGE_PLUGINS_DIR_MISSING	288
XCTEST_UI_TEST_PACKAGE_XCTEST_DIR_MISSING_IN_PLUGINS_DIR	289
XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING	290
XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING_IN_XCTEST_DIR	291
XCTEST_UI_TEST_PACKAGE_CPU_ARCHITECTURE_VALUE_MISSING	292
XCTEST_UI_TEST_PACKAGE_PLATFORM_VALUE_MISSING	294
XCTEST_UI_TEST_PACKAGE_WRONG_PLATFORM_DEVICE_VALUE	295
XCTEST_UI_TEST_PACKAGE_FORM_FACTOR_VALUE_MISSING	297
XCTEST_UI_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING	298
XCTEST_UI_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING	299
XCTEST_UI_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSING	301
XCTEST_UI_TEST_PACKAGE_TEST_EXECUTABLE_VALUE_MISSING	302
Sicurezza	304
Gestione dell'identità e degli accessi	305
Destinatari	305
Autenticazione con identità	305
Come funziona AWS Device Farm con IAM	308
Gestione dell'accesso con policy	314
Esempi di policy basate su identità	316
Risoluzione dei problemi	321
Convalida della conformità	324
Protezione dei dati	325
Crittografia in transito	326
Crittografia a riposo	326
Conservazione dei dati	326
Gestione dei dati	327
Gestione delle chiavi	328
Riservatezza del traffico Internet	328
Resilienza	328
Sicurezza dell'infrastruttura	329
Sicurezza dell'infrastruttura per il test dei dispositivi fisici	329
Sicurezza dell'infrastruttura per i test dei browser desktop	330
Analisi della configurazione e delle vulnerabilità	330
Risposta agli incidenti	331
Registrazione di log e monitoraggio	331
Best practice di sicurezza	331

Limiti	333
Strumenti e plugin	334
Plugin CI Jenkins	334
Fase 1: Installare il plugin	337
Fase 2: Creare un utente IAM	338
Fase 3: Istruzioni per la prima configurazione	339
Fase 4: Usa il plugin	340
Dipendenze	341
Plugin Device Farm Gradle	341
Creazione del plug-in Device Farm Gradle	341
Configurazione del plug-in Device Farm Gradle	342
Generazione di un utente IAM	345
Configurazione dei tipi di test	346
Dipendenze	348
Cronologia dei documenti	349
Glossario per AWS	354
.....	ccclv

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:

- Testing automatizzato delle app sfruttando una vasta gamma di framework di test.
- Accesso remoto ai dispositivi su cui è possibile caricare, eseguire e interagire con le applicazioni in tempo reale.

Note

Device Farm è disponibile solo in us-west-2 regione (Oregon).

Test automatizzato delle app

Device Farm ti consente di caricare i tuoi test o di utilizzare test di compatibilità integrati e senza script. Poiché i test vengono eseguiti in parallelo, i test su più dispositivi iniziano entro pochi minuti.

Una volta completati i test, un rapporto di prova che contiene risultati di alto livello, registri di basso livello, pixel-to-pixelle schermate e i dati sulle prestazioni vengono aggiornati.

Device Farm supporta il test di app Android e iOS native e ibride, incluse quelle create con PhoneGap, Titanium, Xamarin, Unity e altri framework. Supporta l'accesso remoto delle app Android e iOS per i test interattivi. Per ulteriori informazioni sui tipi di test supportati, consulta [Utilizzo dei tipi di test in AWS Device Farm](#).

Interazione con accesso remoto

L'accesso remoto consente di scorrere su, eseguire gesti e interagire con un dispositivo attraverso browser Web in tempo reale. Esistono diverse situazioni in cui l'interazione in tempo reale con un dispositivo risulta utile. Ad esempio, i rappresentanti dell'assistenza clienti possono guidare i clienti all'uso o alla configurazione del dispositivo. Possono anche condurre i clienti all'uso delle app in esecuzione su specifici dispositivi. È possibile installare le app su un dispositivo in esecuzione su una sessione di accesso remoto, per poi riprodurre i problemi del cliente o i bug segnalati.

Durante una sessione di accesso remoto, Device Farm raccoglie 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.

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 [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 [Rapporti 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](#).

session

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, vedere [Configurazione](#).

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](#)
- [Passaggio 2: crea o utilizza un IAM utente nel tuo AWS account](#)
- [Passaggio 3: autorizzare l'IAMutente ad accedere a Device Farm](#)
- [Approfondimenti](#)

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, durante la quale sarà necessario 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 AWS servizi 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 IAM utente 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'IAMutente.

Per ulteriori informazioni, vedere [Creazione di un IAM utente \(AWS Management Console\)](#).

Passaggio 3: autorizzare l'IAMutente ad accedere a Device Farm

Concedi all'IAMutente l'autorizzazione ad accedere a Device Farm. A tale scopo, crea una politica di accesso inIAM, quindi assegna la politica di accesso all'IAMutente, come segue.

Note

L'account o IAM l'utente AWS root che utilizzate per completare i seguenti passaggi deve disporre dell'autorizzazione per creare la seguente IAM politica e allegarla all'IAMutente. Per ulteriori informazioni, consulta l'articolo relativo all'[utilizzo delle policy](#).

1. Create una politica con il seguente JSON corpo. Assegnagli un titolo descrittivo, ad esempio *DeviceFarmAdmin*.

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

Per ulteriori informazioni sulla creazione IAM di politiche, vedere [Creazione IAM di politiche](#) nella Guida per l'IAMutente.

2. Allega la IAM policy che hai creato al tuo nuovo utente. Per ulteriori informazioni su come allegare IAM le politiche agli utenti, consulta [Aggiungere e rimuovere IAM le politiche](#) nella Guida per l'IAMutente.

L'aggiunta della policy fornisce IAM all'utente l'accesso a tutte le azioni e le risorse di Device Farm associate a quell'IAMutente. Per informazioni su come limitare IAM gli utenti a un insieme limitato di azioni e risorse di Device Farm, vedere [Gestione 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 in us-west-2 (Oregon) AWS Regione.

Argomenti

- [Prerequisiti](#)
- [Fase 1: Accesso alla console di](#)
- [Fase 2: Creare un progetto](#)
- [Fase 3: Creare e iniziare una corsa](#)
- [Passaggio 4: Visualizza i risultati della corsa](#)
- [Fasi successive](#)

Prerequisiti

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

- Completa le fasi descritte in [Configurazione](#). Hai bisogno di un AWS account e un AWS Identity and Access Management utente (IAM) con autorizzazione ad accedere a Device Farm.
- Per Android, è necessario disporre di un file .apk (pacchetto di app Android). Per iOS, è necessario disporre di un file .ipa (archivio di app iOS). Il file viene caricato 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 [Utilizzo dei tipi di test 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.

- Accedi 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 Test dei dispositivi mobili, quindi scegli Progetti.
2. Sotto Progetti di test su dispositivi mobili, scegli Nuovo progetto.
3. Sotto Crea progetto, inserisci un Nome del progetto (ad esempio, **MyDemoProject**).
4. Seleziona Create (Crea).

La console apre il Test automatico pagina del tuo progetto appena creato.

Fase 3: Creare e iniziare una corsa

Ora che disponi di un progetto, puoi creare e avviare una sessione. Per ulteriori informazioni, consulta [Esecuzioni](#).

1. Nella pagina Automated tests (Test automatici), selezionare Create a new run (Crea una nuova sessione).
2. Sul Scegli l'applicazione pagina, sotto App per dispositivi mobili, scegli Scegli File, quindi scegli un file Android (.apk) o iOS (.ipa) dal tuo computer. In alternativa, trascina il file dal computer e rilascialo nella console.
3. Inserisci un Nome della corsa, ad esempio **my first test**. Per impostazione predefinita, la console Device Farm utilizza il nome del file.

4. Seleziona Successivo.
 5. SulConfigurapagina, sottoFramework di configurazione e test, scegli uno dei framework di test o delle suite di test integrate. Per ulteriori informazioni su ciascuna opzione, consulta [Tipi e framework di test](#).
 - Se non hai ancora confezionato i test per Device Farm, scegliIntegrato: Fuzzper eseguire una suite di test standard integrata. È possibile mantenere i valori predefiniti perNumero di eventi,Acceleratore per eventi, eSeme del randomizzatore. 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, quindi carica il file che contiene i test.
 6. Seleziona Successivo.
 7. SulSeleziona dispositivipagina, perPool di dispositivi, scegli i migliori dispositivi.
 8. Seleziona Successivo.
 9. Nella pagina Specify device state (Specifica stato dei dispositivi), effettuare una delle seguenti operazioni:
 - Per fornire dati aggiuntivi da utilizzare in Device Farm durante l'esecuzione, inAggiungere dati aggiuntivi, carica un file.zip.
 - Per installare altre app per la corsa, inInstalla altre app, carica i file.apk o.ipa per le app. Per modificare l'ordine di installazione, trascina e rilascia i file.
 - Per attivare le radio Wi-Fi, Bluetooth, GPS o NFC durante la corsa, sottoImposta gli stati della radio, seleziona le caselle di controllo corrispondenti.
-  **Note**

Al momento, l'impostazione dello stato radio del dispositivo è disponibile solo per i test nativi di Android.
- Per testare il comportamento specifico della località durante l'esecuzione, inUbicazione del dispositivo, specificare la preimpostazioneLatitudineeLongitudinecoordinate.
 - Per preimpostare la lingua e la regione del dispositivo per l'esecuzione, inImpostazioni locali del dispositivo, scegli una lingua.
 - Per preimpostare il profilo di rete per la corsa, inProfilo di rete, scegli un profilo curato. Oppure scegliCrea un profilo di reteper crearne uno tuo.

10. Seleziona Successivo.

11. Nella pagina Review and start run (Verifica e avvia sessione), selezionare Confirm and start run (Conferma e avvia sessione).


Device Farm avvia l'esecuzione non appena i dispositivi sono disponibili, in genere entro pochi minuti. Per visualizzare lo stato di esecuzione, suTest automaticipagina del tuo progetto, scegli il nome della tua corsa. Nella pagina di esecuzione, sottoDispositivi, ogni dispositivo inizia con l'icona in sospeso 

tabella dei dispositivi, quindi passa all'icona in

esecuzione 

inizia il 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 del test durante la corsa, suTest automaticipagina del tuo progetto, scegli il nome della tua 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.
- Qualsiasi screenshot acquisito durante la sessione, raggruppato per dispositivo.
- Una sezione per scaricare il risultato dell'analisi.

Per ulteriori informazioni, consulta [Utilizzo dei report di test in Device Farm](#).

Fasi successive

Per ulteriori informazioni su Device Farm, consulta [Concetti](#).

Acquista uno slot per dispositivi in Device Farm

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

Argomenti

- [Acquista slot per dispositivi \(console\)](#)
- [Acquista uno slot per dispositivo \(AWS CLI\)](#)
- [Acquista uno slot per dispositivo \(API\)](#)
- [Annullare lo slot di un dispositivo \(console\)](#)
- [Annulla lo slot di un dispositivo \(AWS CLI\)](#)
- [Annullare lo slot di un dispositivo \(\) API](#)

Acquista slot per dispositivi (console)

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

Purchase and manage device slots

Changes to device slots apply to your entire account and will affect all projects. ×

Automated testing

Automated testing allows you to run built-in or your own tests against devices in parallel with concurrency equal to the number of slots you've purchased. [Learn more](#) >>

Current billing period

You currently have

<input type="text" value="0"/> Android slots	<input type="text" value="0"/> iOS slots
--	--

Next billing period

From August 16, you will have

<input type="text" value="0"/> Android slots	<input type="text" value="0"/> iOS slots
--	--

Remote access

Remote access allows you to manually interact with devices through your browser with the number of concurrent sessions equal to the number of slots you've purchased. [Learn more](#) >>

Current billing period

You currently have

<input type="text" value="0"/> Android slots	<input type="text" value="0"/> iOS slots
--	--

Next billing period

From August 16, you will have

<input type="text" value="0"/> Android slots	<input type="text" value="0"/> iOS slots
--	--

Save

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

Confirm purchase



- **Automated Testing Android slot** will be added to your account and [redacted] will be immediately added to your [redacted] bill.
- In [redacted], you will have **Remote Access Android slot**, **Automated Testing Android slot**, **Automated Testing iOS slot** and **Remote Access iOS slot** and [redacted] will be added to your recurring monthly bill.

Cancel

Confirm

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"
    }
  ]
}

```

```
        "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 [Riferimento AWS CLI](#).

Acquista uno slot per dispositivo (API)

1. Chiama l'[GetAccountSettings](#) operazione per elencare le impostazioni del tuo account.
2. Chiama l'[ListOfferings](#) operazione per elencare le offerte di slot per dispositivi disponibili.
3. Chiama l'[ListOfferingPromotions](#) operazione 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'[PurchaseOffering](#) operazione per acquistare un'offerta.

5. Chiama l'[GetOfferingStatus](#) operazione per conoscere lo stato dell'offerta.

Per questa funzionalità sono disponibili anche i [ListOfferingTransactions](#) comandi [RenewOffering](#) and.

Per informazioni sull'utilizzo di Device Farm API, vedere [Automazione di Device Farm](#).

Annullare lo slot di un dispositivo (console)

1. Accedere alla console Device Farm all'[indirizzo https://console.aws.amazon.com/devicefarm](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. Selezionare Salva. Apparirà una finestra di conferma della modifica. Controlla le informazioni, quindi scegli Conferma per completare la transazione.

Annula 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'[RenewOffering](#) operazione per modificare la quantità di dispositivi nel tuo account.

AWSConcetti di Device Farm

Questa sezione descrive importanti concetti di Device Farm.

- [Supporto per dispositivi in AWS Device Farm](#)
- [Ambienti di test](#)
- [Esecuzioni](#)
- [App](#)
- [Rapporti in AWS Device Farm](#)
- [Sessioni](#)

Per ulteriori informazioni sui tipi di test supportati in Device Farm, vedere [Utilizzo dei tipi di test 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 [Elenco dispositivi](#).

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 [Utilizzo di 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 hanno una connessione Wi-Fi a Internet. Non dispongono di connessioni con l'operatore e non possono effettuare telefonate o inviare SMS messaggi.

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 è installato sui dispositivi che lo supportano, ma tali dispositivi non hanno un account Google attivo.

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](#)
- [Ambiente di test personalizzato](#)

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 [Lavorare con ambienti di test personalizzati](#).

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

Per ulteriori informazioni, consulta [Caricamento di una specifica di test personalizzata utilizzando and](#). AWS CLI [Crea 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, impostazioni locali, stati radio (come Bluetooth e Wi-Fi) GPSNFC, dati aggiuntivi (contenuti in un file.zip) e app ausiliarie (app che devono essere installate prima dell'app da testare).

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 [Imposta il 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, consulta [Crea un'esecuzione di test in Device Farm](#) e [Utilizzo delle esecuzioni di test 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 UUID di Device Farm 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.

Rapporti in AWS Device Farm

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

Argomenti

- [Conservazione dei report](#)
- [Componenti dei report](#)
- [Registra i report di accesso](#)
- [Attività comuni per i report](#)

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 [Utilizzo dei report di test in Device Farm](#).

Sessioni in AWS Device Farm

Puoi utilizzare Device Farm per eseguire test interattivi di app Android e iOS tramite sessioni di accesso remoto in un browser web. Test interattivi di questo tipo aiutano gli addetti al supporto telefonico ad assistere i clienti con il loro problema fornendo istruzioni dettagliate. Gli sviluppatori possono riprodurre un problema su un determinato dispositivo per isolare possibili cause. Puoi utilizzare sessioni remote per condurre test di usabilità con i tuoi clienti target.

Argomenti

- [Dispositivi supportati per l'accesso remoto](#)
- [Conservazione dei file di sessione](#)
- [App di strumentazione](#)
- [Riassegnare la firma delle app nelle sessioni](#)
- [App offuscate nelle sessioni](#)

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 rifica 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 misure antipirateria o antimanomissione relative a prodotti come DexGuard per dispositivi Android.

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 si utilizza DexGuard con misure antipirateria, Device Farm non può firmare nuovamente l'app.

Lavorare con progetti in AWS Device Farm

Un progetto in Device Farm rappresenta uno spazio di lavoro logico in Device Farm che contiene esecuzioni, una 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 usare la console AWS Device Farm, AWS Command Line Interface (AWS CLI) o l'API AWS Device Farm per lavorare con i progetti.

Argomenti

- [Crea un progetto in AWS Device Farm](#)
- [Visualizza l'elenco dei progetti in AWS Device Farm](#)

Crea un progetto in AWS Device Farm

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

Prerequisiti

- Completa le fasi descritte in [Configurazione](#).

Crea un progetto (console)

1. Accedi alla console Device Farm all'indirizzo <https://console.aws.amazon.com/devicefarm>.
2. Nel pannello di navigazione di Device Farm, scegli Test dei dispositivi mobili, quindi scegli Progetti.
3. Scegliere New project (Nuovo progetto).
4. Inserisci un nome per il tuo progetto, quindi scegli Invia.
5. Per specificare le impostazioni per il progetto, scegliere Project settings (Impostazioni progetto). Queste impostazioni includono il timeout predefinito per le esecuzioni di test. Una volta applicate, le impostazioni vengono utilizzate in tutte le esecuzioni di test del progetto. Per ulteriori informazioni, consulta [Imposta il timeout di esecuzione per le esecuzioni di test in AWS Device Farm](#).

Crea un progetto (AWS CLI)

- Eseguire `create-project` specificando il nome del progetto.

Esempio:

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

La risposta dell'AWS CLI 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 [Riferimento AWS CLI](#).

Crea un progetto (API)

- Chiamata dell'API [CreateProject](#).

Per informazioni sull'utilizzo dell'API Device Farm, vedere [Automazione di Device Farm](#).

Visualizza l'elenco dei progetti in AWS Device Farm

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

Argomenti

- [Prerequisiti](#)
- [Visualizza l'elenco dei progetti \(console\)](#)
- [Visualizza l'elenco dei progetti \(AWS CLI\)](#)
- [Visualizza l'elenco dei progetti \(API\)](#)

Prerequisiti

- Crea almeno un progetto in Device Farm. Segui le istruzioni riportate in [Crea un progetto in AWS Device Farm](#), poi torna a questa pagina.

Visualizza l'elenco dei progetti (console)

1. Accedi 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 su dispositivi mobili, nel menu di navigazione di Device Farm, scegli **Test dei dispositivi mobili**, quindi scegli **Progetti**.
 - Per i progetti di test dei browser desktop, nel menu di navigazione di Device Farm, scegli **Test del browser desktop**, 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, vedere [Riferimento AWS CLI](#).

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, consulta [Automazione di Device Farm](#).

Utilizzo delle esecuzioni di test 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](#).

È possibile utilizzare la console AWS Device Farm, AWS Command Line Interface (AWS CLI) o AWS Device Farm API per lavorare con le esecuzioni.

Argomenti

- [Crea un'esecuzione di test in Device Farm](#)
- [Imposta il timeout di esecuzione per le esecuzioni di test in AWS Device Farm](#)
- [Simula le connessioni e le condizioni di rete per il funzionamento di AWS Device Farm](#)
- [Interrompere un'esecuzione in AWS Device Farm](#)
- [Visualizza un elenco di esecuzioni in AWS Device Farm](#)
- [Creare un pool di dispositivi in AWS Device Farm](#)
- [Analisi dei risultati in AWS Device Farm](#)

Crea un'esecuzione di test in Device Farm

È possibile utilizzare la console Device Farm o Device Farm API per creare un'esecuzione di test. AWS CLI 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](#)
- [Crea un'esecuzione di test \(console\)](#)
- [Crea un test run \(AWS CLI\)](#)
- [Creare un test run \(API\)](#)
- [Passaggi successivi](#)

Prerequisiti

È necessario disporre di un progetto in Device Farm. Segui le istruzioni riportate in [Crea 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](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 selezionare Create a new run (Crea una nuova sessione).
5. Nella pagina Scegli l'applicazione, scegli App per dispositivi mobili o App Web.

Step 1
Choose application

Step 2
Configure

Step 3
Select devices

Step 4
Specify device state

Step 5
Review and start run

Choose application

Mobile App | Web App

Upload an Android app as a .apk. Upload an iOS app as a .ipa. Be sure to build for 'iOS device'. No instrumentation or provisioning required

Choose File or drop file here or Select a recent upload

Cancel Next step

6. Caricare il file dell'applicazione. È inoltre possibile trascinare il file oppure selezionare un caricamento recente. Se si sta caricando un'app iOS, assicurarsi di selezionare iOS device (dispositivo iOS), invece di un simulatore.
7. (Opzionale) In Run name (Nome sessione), immettere un nome. Per impostazione predefinita, Device Farm utilizza il nome del file dell'app.
8. Seleziona Successivo.
9. Nella pagina Configure (Configura), selezionare una delle suite 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 [Utilizzo dei tipi di test in AWS Device Farm](#).

10. Se non hai scelto Built-in: Fuzz, seleziona Scegli file, quindi cerca e scegli il file che contiene i 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](#).
12. Se stai utilizzando l'ambiente di test standard, passa alla fase 13. Se utilizzi un ambiente di test personalizzato con il YAML file delle specifiche di test predefinito, vai al passaggio 13.
 - a. Se desideri modificare la specifica di test predefinita in un ambiente di test personalizzato, scegli Modifica per aggiornare la specifica predefinita. YAML
 - b. Se hai modificato la specifica di test, scegli Salva come nuova per aggiornarla.
13. Se si desidera configurare la registrazione video o le prestazioni delle opzioni di acquisizione di dati, selezionare Advanced Configuration (Configurazione avanzata).
 - a. Seleziona Abilita la registrazione video per registrare video durante il test.
 - b. Seleziona Abilita l'acquisizione dei dati sulle prestazioni dell'app per acquisire i dati sulle prestazioni dal dispositivo.

Note

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

14. Seleziona Successivo.
15. Nella pagina Select devices (Seleziona dispositivo), eseguire 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 [Creare 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.

Per ulteriori informazioni, consulta la pagina [Supporto per dispositivi in AWS Device Farm](#).

16. Seleziona Next (Successivo).
17. Nella pagina Specify device state (Specifica stato dispositivo):
 - 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 Wi-Fi, Bluetooth o NFC è abilitato durante la corsaGPS, accanto a Imposta gli stati della radio, seleziona 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.
18. Seleziona Successivo.
19. Nella pagina Rivedi e avvia l'esecuzione, puoi specificare il timeout di esecuzione per l'esecuzione del test. Se si stanno utilizzando slot di test illimitati, controllare che Run on unmetered slots (Esegui su slot non misurati) sia selezionato.
20. Inserire un valore o utilizzare la barra di scorrimento per modificare il timeout di esecuzione. Per ulteriori informazioni, consulta [Imposta il timeout di esecuzione per le esecuzioni di test in AWS Device Farm](#).
21. Selezionare Confirm and start run (Controlla e avvia sessione).

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 si trasforma in un'icona del risultato del test.

Se desideri interrompere l'esecuzione del test, consulta [Interrompere 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](#)
- [Passaggio 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, esegui `list-projects`. Se non disponi ancora di un progetto, consulta [Crea 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. Scegli un progetto da associare all'esecuzione del test e prendi nota del relativo 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, esegui `list-device-pools`, specificando il tuo progetto ARN.

Esempio:

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

La risposta include i pool di dispositivi Device Farm integrati, ad esempio Top 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. Scegliete un pool di dispositivi e prendetene notaARN.

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 caricamento predefinito di Amazon Simple Storage Service (Amazon S3), devi: URL

- Il tuo progetto. ARN
- 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 il caricamento dell'app ARN e un predefinito. URL

```
{
```

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

2. Prendi nota dell'app caricata ARN e di quella predefinitaURL.
3. Carica il file dell'app utilizzando Amazon S3 presigned. URL 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'app, esegui get-upload e specifica lo stato del caricamento ARN dell'app.

```
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 caricamento predefinito di Amazon S3URL, 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 il caricamento del pacchetto di test ARN e un pacchetto predefinito. URL

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

2. Prendi nota del ARN pacchetto di test caricato e del pacchetto URL prefirato.
3. Carica il file del pacchetto degli script di test utilizzando il pacchetto Amazon URL S3 prefirato. 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 di script di test, esegui `get-upload` e specifica il caricamento del pacchetto ARN di test dal passaggio 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}"
  }
}
```

Passaggio 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](#).

1. Per trovare il caricamento ARN per le specifiche di test predefinite, esegui list-uploads e specifica il progetto. ARN

```
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. Prendi nota del suo caricamentoARN.
3. Per scaricare le specifiche di test predefinite, esegui get-upload e specifica il caricamentoARN.

Esempio:

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

La risposta contiene un predefinito URL in cui è possibile scaricare le specifiche di test predefinite.

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 delle specifiche di test personalizzate, esegui, specificando il nome della specifica di test create-upload, il tipo di specifica di test e il progetto. ARN

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 il caricamento e la prefirma delle specifiche di test: ARN URL

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

7. Prendi nota del caricamento e del ARN prefirmato delle specifiche per il test. URL
8. Carica il file delle specifiche di test utilizzando Amazon URL S3 presigned. 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 delle specifiche di test, esegui get-upload e specifica il caricamento. ARN

```
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 le specifiche di test personalizzate, esegui `update-upload`, specificando il caricamento ARN 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, esegui `schedule-run`, specificando:

- Il progetto ARN della [fase 1](#).
- Il pool di dispositivi ARN del [passaggio 2](#).
- Il caricamento dell'app ARN dal [passaggio 3](#).
- Il caricamento del pacchetto di test ARN dal [passaggio 4](#).

Se stai eseguendo test in un ambiente di test personalizzato, hai bisogno anche delle specifiche di test ARN del [passaggio 5](#).

Per pianificare una sessione in un ambiente di test standard

- Esegui `schedule-run`, specificando le informazioni sul progetto ARN, sul pool di dispositivi ARN, sul caricamento ARN dell'applicazione e sul 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'esecuzione ARN che è possibile utilizzare per verificare lo stato dell'esecuzione del 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

- Usa il `get-run` comando e specifica l'esecuzioneARN:

```
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, vedere [Riferimento AWS CLI](#).

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 [ScheduleRun](#)API:

- Un progettoARN. Consulta [Crea un progetto \(API\)](#) e [CreateProject](#).
- Caricamento di un'applicazioneARN. Per informazioni, consulta [CreateUpload](#).
- Caricamento di un pacchetto di provaARN. Per informazioni, consulta [CreateUpload](#).
- Un pool di dispositiviARN. Consulta [Creare un pool di dispositivi](#) e [CreateDevicePool](#).

Note

Se esegui test in un ambiente di test personalizzato, devi anche caricare ARN le specifiche di test. Per ulteriori informazioni, consulta [Passaggio 5: \(Facoltativo\) Carica le specifiche di test personalizzate](#) e [CreateUpload](#).

Per informazioni sull'utilizzo di Device FarmAPI, vedere [Automazione di Device Farm](#).

Passaggi successivi

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 [Rapporti in AWS Device Farm](#).

Per utilizzare il rapporto, segui le istruzioni riportate in [Utilizzo dei report di test in Device Farm](#).

Imposta il 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. È possibile utilizzare la console AWS Device Farm o AWS Device Farm API 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 sezione [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 [Crea 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](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. Selezionare 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](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).

Simula le connessioni e le condizioni di rete per il funzionamento 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, FireOS e web in Device Farm. Ad esempio, è possibile testare la tua applicazione in condizioni di rete non ottimali.

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 utilizzi il network shaping, puoi 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](#)
- [Crea un profilo di rete](#)
- [Modifica le condizioni della rete durante il test](#)

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 [Crea 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.

Network profile

Select a pre-defined network profile or create a new one by clicking the button on the right.

Full ▼

Create network profile

6. Seleziona 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.

Create network profile ✕

Name

Description - optional

Uplink bandwidth (bps)
Data throughput rate in bits per second as a number from 0 to 105487600.

Downlink bandwidth (bps)
Data throughput rate in bits per second as a number from 0 to 105487600.

Uplink delay (ms)
Delay time for all packets to destination in milliseconds as a number from 0 to 2000.

Downlink delay (ms)
Delay time for all packets to destination in milliseconds as a number from 0 to 2000.

Uplink jitter (ms)
Time variation in the delay of received packets in milliseconds as a number from 0 to 2000.

Downlink jitter (ms)
Time variation in the delay of received packets in milliseconds as a number from 0 to 2000.

Uplink loss (%)
Proportion of transmitted packets that fail to arrive from 0 to 100 percent.

Downlink loss (%)
Proportion of received packets that fail to arrive from 0 to 100 percent.

2. Immettere un nome e le impostazioni per il proprio profilo di rete.
3. Scegli 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).

General						
Device pools						
Network profiles						
Uploads						
Network profiles						<input type="button" value="Refresh"/> <input type="button" value="Edit"/> <input type="button" value="Delete"/> <input type="button" value="Create network profile"/>
Name	Bandwidth (bps)	Delay (ms)	Jitter (ms)	Loss (%)	Description	
<input type="radio"/>	104857600	1048576	0	0	-	
<input type="radio"/>	104857600	1048576	0	0	-	
<input type="radio"/>	104857600	1048576	0	0	-	

Modifica le condizioni della rete durante il test

Puoi chiamare un server 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](#)

Interrompere 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.

È possibile utilizzare la console Device Farm o API interrompere una corsa. AWS CLI

Argomenti

- [Interrompi una corsa \(console\)](#)
- [Interrompi una corsa \(\)AWS CLI](#)
- [Interrompere una corsa \(API\)](#)

Interrompi una corsa (console)

1. Accedere alla console Device Farm all'[indirizzo https://console.aws.amazon.com/devicefarm](https://console.aws.amazon.com/devicefarm).
2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.
3. Scegli il progetto in cui eseguire 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.

aws-devicefarm-sample-app.apk Scheduled at: Thu Jul 15 2021 19:03:03 GMT-0700 (Pacific Daylight Time)

Run ARN: Stop run

No recent tests

■ Passed ■ Failed ■ Errored ■ Warned ■ Stopped ■ Skipped

🔔 Your app is currently being tested. Results will appear here as tests complete.

0 out of 5 devices completed 0%

Devices | Unique problems | Screenshots | Parsing result

Devices

🔍 Find device by status, device name, or OS < 1 > ⌂

Status	Device	OS	Test Results	Total Minutes
🔄 Running	Google Pixel 4 XL (Unlocked)	10	Passed: 0, errored: 0, failed: 0	00:00:00
🔄 Running	Samsung Galaxy S20 (Unlocked)	10	Passed: 0, errored: 0, failed: 0	00:00:00

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.

Devices					
Status	Device	OS	Test Results	Total Minutes	
⊖ Stopped	Google Pixel 4 XL (Unlocked)	10	Passed: 2, errored: 0, failed: 0	00:01:37	
⊖ Stopped	Samsung Galaxy S20 (Unlocked)	10	Passed: 2, errored: 0, failed: 0	00:02:04	
⊖ Stopped	Samsung Galaxy S20 ULTRA (Unlocked)	10	Passed: 2, errored: 0, failed: 0	00:01:57	
⊖ Failed	Samsung Galaxy S9 (Unlocked)	9	Passed: 2, errored: 0, failed: 1	00:01:36	
⊖ Stopped	Samsung Galaxy Tab S4	8.1.0	Passed: 2, errored: 0, failed: 0	00:01:31	

Interrompi una corsa ()AWS CLI

È possibile eseguire il comando seguente per interrompere l'esecuzione del test specificata, dove *myARN* è 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 il ARN risultato della tua corsa, usa il `list-runs` comando. 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, vedere [Riferimento AWS CLI](#).

Interrompere una corsa (API)

- Richiama l'[StopRun](#) operazione per l'esecuzione del test.

Per informazioni sull'utilizzo di Device FarmAPI, vedere [Automazione di Device Farm](#).

Visualizza un elenco di esecuzioni in AWS Device Farm

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

Argomenti

- [Visualizza un elenco di esecuzioni \(console\)](#)
- [Visualizza un elenco di esecuzioni \(AWS CLI\)](#)
- [Visualizza 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](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, vedere [Riferimento AWS CLI](#).

Visualizza un elenco di esecuzioni (API)

- Chiama il [ListRuns](#)API.

Per visualizzare informazioni su una singola corsa, chiama il [GetRun](#)API.

Per informazioni su Device FarmAPI, vedere [Automazione di Device Farm](#).

Creare un pool di dispositivi in AWS Device Farm

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

Argomenti

- [Prerequisiti](#)
- [Crea un pool di dispositivi \(console\)](#)
- [Crea un pool di dispositivi \(\)AWS CLI](#)
- [Creare un pool di dispositivi \(API\)](#)

Prerequisiti

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

Creare 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 (EQUALSCONTAINS, ecc.) per includere i dispositivi in base alla query. Ad esempio, puoi scegliere *Availability EQUALS AVAILABLE* per includere i dispositivi che attualmente hanno lo Available stato.
- 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 ANDROIDe IOS. Analogamente, se scegli Form Factor for Field, le uniche selezioni disponibili sono PHONEe. 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 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, vedere [Riferimento AWS CLI](#).

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.

- Chiama il [CreateDevicePool](#) API.

Per informazioni sull'utilizzo di Device Farm API, vedere [Automazione di Device Farm](#).

Analisi dei risultati in AWS Device Farm

Nell'ambiente di test standard, puoi utilizzare la console Device Farm per visualizzare i report di ogni test durante l'esecuzione del test.

Device Farm raccoglie anche altri elementi come file, log e immagini che è possibile scaricare al termine del test.

Argomenti

- [Utilizzo dei report di test in Device Farm](#)
- [Lavorare con gli artefatti in Device Farm](#)

Utilizzo dei report di test in Device Farm

Usa la console Device Farm per visualizzare i report dei test. Per ulteriori informazioni, consulta [Rapporti in AWS Device Farm](#).

Argomenti

- [Prerequisiti](#)
- [Comprensione dei risultati dei test](#)
- [Visualizzazione dei report](#)

Prerequisiti

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

1. Per creare un'esecuzione, consulta [Crea 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'inizio dell'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 si trasforma in un'icona del risultato del test. Per ulteriori informazioni, consulta [Comprensione dei risultati dei test](#).

icon

Comprensione dei risultati dei test

La console Device Farm mostra icone che consentono di valutare rapidamente lo stato dell'esecuzione del test completata.

Argomenti

- [Segnalazione dei risultati di un singolo test](#)
- [Segnalazione dei risultati di più test](#)

Segnalazione dei risultati di un singolo test

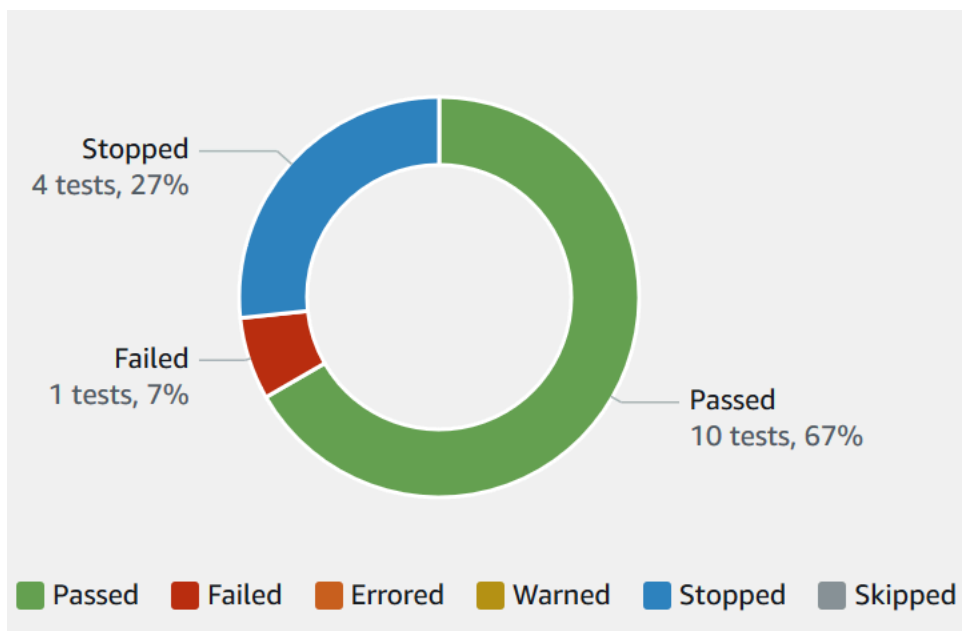
Per i report che descrivono un singolo test, Device Farm visualizza un'icona:

Descrizione	Icon
Il test ha avuto esito positivo.	
Il test ha avuto esito negativo.	

Descrizione	Icon
Device Farm ha saltato il test.	⊗
Il test si è interrotto.	⊖
Device Farm ha restituito un avviso.	⚠
Device Farm ha restituito un errore.	⊖

Segnalazione dei risultati di più test

Se si sceglie una corsa completata, Device Farm visualizza un grafico riassuntivo dei risultati del test.



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

I grafici sono sempre codificati a colori ed etichettati.

Visualizzazione dei 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 i dati sulle prestazioni per un problema, un dispositivo, una suite o un test in un report](#)
- [Visualizza le informazioni di registro relative a un problema, un dispositivo, una suite o un test in un report](#)


Visualizza la pagina di riepilogo dell'esecuzione del test

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

Devices	Unique problems	Screenshots	Parsing result	
Devices <input type="text" value="Find device by status, device name, or OS"/> < 1 > 				
Status	Device	OS	Test Results	Total Minutes
✓ Passed	Google Pixel 4 XL (Unlocked)	10	Passed: 3, errored: 0, failed: 0	00:02:36
✓ Passed	Samsung Galaxy S20 (Unlocked)	10	Passed: 3, errored: 0, failed: 0	00:02:34
✗ Failed	Samsung Galaxy S20 ULTRA (Unlocked)	10	Passed: 2, errored: 0, failed: 1	00:02:25
✓ Passed	Samsung Galaxy S9 (Unlocked)	9	Passed: 3, errored: 0, failed: 0	00:02:46
✓ Passed	Samsung Galaxy Tab S4	8.1.0	Passed: 3, errored: 0, failed: 0	00:03:13

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 [Segnalazione dei risultati 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 report](#).

La scheda Prestazioni mostra informazioni su tutti i dati sulle prestazioni generati da Device Farm durante il test. Per visualizzare questi dati sulle prestazioni, segui le istruzioni contenute in [Visualizza i dati sulle prestazioni per un problema, un dispositivo, una suite o un test in un report](#).

La scheda File visualizza 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 [Segnalazione dei risultati di più test](#).

Per visualizzare i risultati completi per suite, segui le istruzioni riportate in [Visualizza 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 report](#).

La sezione Prestazioni visualizza informazioni su tutti i dati sulle prestazioni generati da Device Farm per il dispositivo durante l'esecuzione. Per visualizzare questi dati sulle prestazioni, segui le istruzioni contenute in [Visualizza i dati sulle prestazioni per un problema, un dispositivo, una suite o un test in un report](#).

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 [Segnalazione dei risultati di più test](#).

Per visualizzare i risultati completi dei test, segui le istruzioni riportate in [Visualizzazione 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 report](#).

La sezione Performance mostra informazioni su tutti i dati sulle prestazioni generati da Device Farm durante l'esecuzione della suite. Per visualizzare questi dati sulle prestazioni, segui le istruzioni contenute in [Visualizza i dati sulle prestazioni per un problema, un dispositivo, una suite o un test in un report](#).

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 [Segnalazione dei risultati 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 report](#).

La scheda Prestazioni mostra informazioni su tutti i dati sulle prestazioni generati da Device Farm durante il test. Per visualizzare questi dati sulle prestazioni, segui le istruzioni contenute in [Visualizza i dati sulle prestazioni per un problema, un dispositivo, una suite o un test in un report](#).

La scheda File visualizza 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 dati sulle prestazioni per un problema, un dispositivo, una suite o un test in un report

Note

Device Farm raccoglie i dati sulle prestazioni del dispositivo solo per gli host di test Android legacy che non utilizzano l'host di `amazon_linux_2` test più recente. Questa funzionalità non è supportata su iOS.

La scheda Prestazioni visualizza le seguenti informazioni:

- Il CPUgrafico mostra la percentuale di CPU utilizzo dell'app su un singolo core durante il problema, il dispositivo, la suite o il test selezionato (lungo l'asse verticale) nel tempo (lungo l'asse orizzontale).

L'asse verticale viene espresso in percentuali da 0% al valore massimo registrato.

La percentuale può superare il 100% nel caso in cui l'applicazione utilizzi più di un core. Ad esempio, se tre core registrano un utilizzo del 60%, questa percentuale viene mostrata come 180%.

- Il grafico della memoria mostra il numero di MB utilizzati dall'app durante il problema, il dispositivo, la suite o il test selezionato (lungo l'asse verticale) nel tempo (lungo l'asse orizzontale).

L'asse verticale viene espresso in MB, da 0 al numero massimo di MB registrato.

- Il grafico Threads (Thread) mostra il numero di thread utilizzati con riferimento al problema, al dispositivo, alla suite o al test selezionato (lungo l'asse verticale) nel tempo (lungo l'asse orizzontale).

L'asse verticale è espresso in numero di thread, da zero thread al numero massimo di thread registrati.

In tutti i casi, l'asse orizzontale è indicato in secondi dall'inizio e dalla fine dell'esecuzione con riferimento al problema, al dispositivo, alla suite o al test selezionato.

Per visualizzare informazioni per un determinato punto dati, sospendi il grafico al secondo desiderato lungo l'asse orizzontale.

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

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 **MM:SS.SSS** formato, dove **M** rappresenta i minuti e **S** rappresenta i secondi.
- PID rappresenta l'identificatore di processo (PID) che ha creato la voce di registro. Tutte le voci di registro create da un'app su un dispositivo sono uguali 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

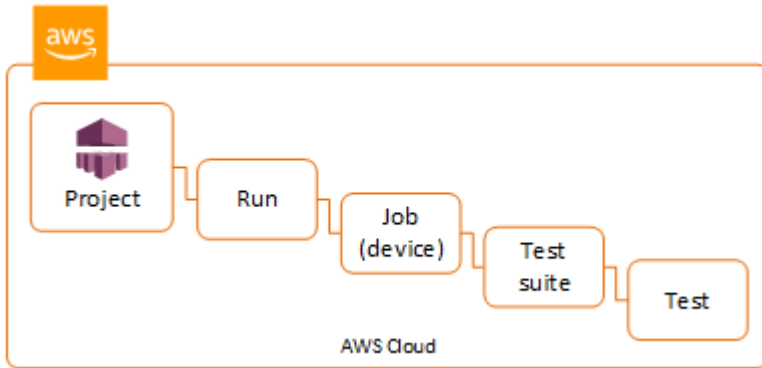
- Esegui il debug
 - Emergenza
 - Errore
 - Errore
 - Failed (Non riuscito)
 - Informazioni
 - Interno
 - Comunicazione
 - Superato
 - Saltato
 - Arrestate
 - Modalità dettagliata
 - Notifica
 - Attenzione
- Tag indica metadati arbitrari per la voce di log. Ad esempio, il logcat Android può servirsene 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 del dispositivo, incluse tutte le suite e i test eseguiti, scegli Scarica registri.

Lavorare con gli 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 [Utilizzo dei report di test in Device Farm](#).

Log

Output da ciascun test nella sessione.

Screenshot

Immagini di schermata registrate per ogni test nella sessione.

Utilizzo di 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 [Utilizzo di artefatti in un ambiente di test personalizzato](#).

Utilizzo degli artefatti ()AWS CLI

È possibile utilizzare il AWS CLI per elencare gli artefatti del test eseguito.

Argomenti

- [Fase 1: Ottieni i nomi delle tue risorse Amazon \(ARN\)](#)
- [Fase 2: Elenca i tuoi artefatti](#)
- [Passaggio 3: scarica i tuoi artefatti](#)

Fase 1: Ottieni i nomi delle tue risorse Amazon (ARN)

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

AWS CLI Comando List	Richiesto ARN
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 un test ARN, esegui list-tests utilizzando la tua suite di test ARN come parametro di input.

Esempio:

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

La risposta include un test ARN 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 ha un codice che consente di scaricare il file. URL

- Chiama list-artifacts specificando un'esecuzione, un lavoro, una suite di test o un test. ARN Specificare un tipo di FILELOG, oSCREENSHOT.

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

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

La risposta contiene un download URL 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 il tuo artefatto utilizzando il passaggio precedente URL. Questo esempio utilizza curl per scaricare un file di output Android Appium Java:

```

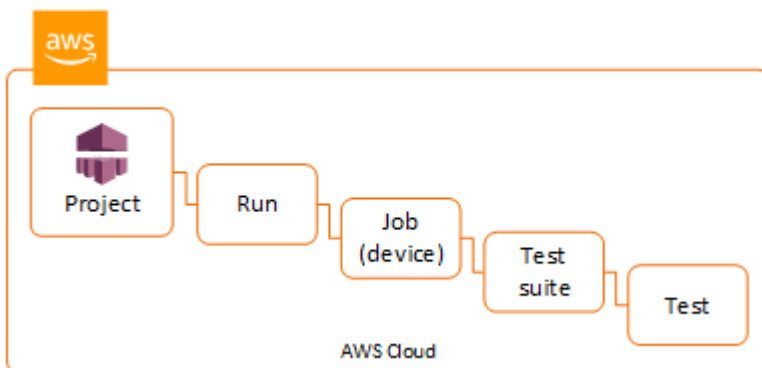
curl "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/ExampleURL"
> MyArtifactName.txt

```

Utilizzo degli artefatti () API

Il API [ListArtifacts](#) metodo Device Farm restituisce un elenco di elementi, come file, schermate e registri. Ogni elemento è dotato di un URL codice che consente di scaricare il file.

Utilizzo 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 dell'esecuzione dei comandi nel file delle specifiche di test. YAML

Artefatti personalizzati

Un file compresso che contiene gli artefatti della sessione di test. È configurato nella sezione `artifacts` del file delle specifiche di test. YAML

Script della shell di specifica di test

Un file di script di shell intermedio creato dal tuo file. YAML Poiché viene utilizzato nell'esecuzione del test, il file di script di shell può essere utilizzato per il debug del file. YAML

File di specifica di test

Il YAML file utilizzato nell'esecuzione del test.

Per ulteriori informazioni, consulta [Lavorare con gli artefatti in Device Farm](#).

Taggare le risorse di AWS Device Farm

AWS Device Farm funziona con AWS API di etichettatura dei gruppi di risorse. 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.

Puoi 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

- [Assegnazione di tag alle risorse](#)
- [Ricerca di risorse per tag](#)
- [Rimozione dei tag dalle risorse](#)

Assegnazione 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 prende un elenco di ARN dai 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 Python applicherà tag a una serie di ARN di progetto con il tag `build-config` e 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 `resourcegroupstaggingapi`. 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 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 di test del browser desktop Device Farm (`devicefarm:testgrid-sessionresources`) con il tag `stackche` che hanno 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"])
```

Utilizzo dei tipi di test in AWS Device Farm

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

Framework di test

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

Framework di test delle applicazioni Android

- [Lavorare con Appium e AWS Device Farm](#)
- [Utilizzo della strumentazione per Android e AWS Device Farm](#)

Framework di test delle applicazioni iOS

- [Lavorare con Appium e AWS Device Farm](#)
- [Lavorare con XCTest per iOS e AWS Device Farm](#)
- [XCTestInterfaccia 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 [Lavorare con Appium e AWS 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 [Lavorare con ambienti di test personalizzati](#).

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](#).

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 \(Android e iOS\)](#)

Lavorare con Appium e AWS Device Farm

Questa sezione descrive come configurare, impacchettare e caricare i test Appium su Device Farm. Appium è uno strumento open source per l'automazione di applicazioni web native e mobili. Per ulteriori informazioni, vedere [Introduzione ad Appium sul sito Web di Appium](#).

Per un'app di esempio e collegamenti ai test di lavoro, consulta [Device Farm Sample App per Android](#) e [Device Farm Sample App per iOS](#) su GitHub.

Supporto versione

Il supporto per vari framework e linguaggi di programmazione dipende dal linguaggio utilizzato.

Device Farm supporta tutte le versioni del server Appium 1.x e 2.x. Per Android, puoi scegliere qualsiasi versione principale di Appium con `devicefarm-cli`. Ad esempio, per utilizzare la versione 2 del server Appium, aggiungi questi comandi al file delle specifiche di test: YAML

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

Per iOS, puoi scegliere versioni specifiche di Appium con i comandi `avm` o `npm`. Ad esempio, per utilizzare il `avm` comando per impostare la versione del server Appium sulla 2.1.2, aggiungi questi comandi al file delle specifiche di test: YAML

```
phases:
  install:
    commands:
```

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

Utilizzando il npm comando per utilizzare l'ultima versione di Appium 2, aggiungi questi comandi al file delle specifiche di test: YAML

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

[Per ulteriori informazioni devicefarm-cli o per qualsiasi altro CLI comando, consulta il riferimento. AWS CLI](#)

Per utilizzare tutte le funzionalità del framework, come le annotazioni, scegli un ambiente di test personalizzato e usa la console AWS CLI o la Device Farm console per caricare una specifica di test personalizzata.

Argomenti

- [Configura il tuo pacchetto di test Appium](#)
- [Crea un file di pacchetto di test compresso](#)
- [Carica il tuo pacchetto di test su Device Farm](#)
- [Acquisisci schermate dei tuoi test \(opzionale\)](#)

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 JAR file:

```
<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 JAR file.

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

```
<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 JAR file.

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 XML assieme in `src/main/assembly/zip.xml`

Di seguito XML è riportata 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 JAR file:

```
<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 JAR file.

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

```
<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 JAR file.

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 XML assieme in `src/main/assembly/zip.xml`

Di seguito XML è riportata 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 \(npm\)](#)

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:

- [Gestore delle versioni di Ruby \(\) 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. Installa le chiavi richieste e Ruby. RVM Per istruzioni, consulta [Installazione RVM](#) sul RVM sito Web.

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

Note

RVM viene caricato come funzione solo per la 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 la struttura del ZIP file in modo implicito. 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 utilizzare la console Device Farm per caricare i test.

1. Accedere alla console Device Farm all'[indirizzo https://console.aws.amazon.com/devicefarm](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 selezionare Create a new run (Crea una nuova sessione).
5. Per i test nativi di Android e iOS

Nella pagina Scegli l'applicazione, scegli App per dispositivi mobili, quindi seleziona Scegli file 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

Nella pagina Scegli l'applicazione, scegli App Web.

6. Dare al test un nome appropriato. Tale nome può contenere qualsiasi combinazione di spazi o punteggiatura.
7. Seleziona Successivo.
8. Nella pagina Configura, nella sezione Setup test framework, scegli Appium **Language**, quindi Scegli File.
9. 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](#).
10. Scegli Esegui il test in un ambiente personalizzato. Questo ambiente di esecuzione consente il pieno controllo sulla configurazione, lo smontaggio e l'invocazione dei test, nonché sulla scelta di versioni specifiche dei runtime e del server Appium. È possibile configurare l'ambiente personalizzato tramite il file delle specifiche di test. Per ulteriori informazioni, consulta [Lavorare con ambienti di test personalizzati in AWS Device Farm](#).
11. Scegli Avanti, quindi segui le istruzioni per selezionare i dispositivi e avviare l'esecuzione. Per ulteriori informazioni, consulta [Crea 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 schermate nei test di Appium, consulta [Take Screenshot](#) nella documentazione di Appium. API

Utilizzo dei test Android in AWS Device Farm

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

Framework di test delle applicazioni Android

Sono disponibili i seguenti test per i dispositivi Android.

- [Lavorare con Appium e AWS Device Farm](#)
- [Utilizzo della strumentazione per Android e AWS Device Farm](#)

Tipi di test integrati per Android

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

- [Integrato: fuzz \(Android e iOS\)](#)

Utilizzo della 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.

Argomenti

- [Che cos'è la strumentazione?](#)
- [Carica i tuoi test di strumentazione Android](#)
- [Acquisizione di schermate nei test della strumentazione Android](#)
- [Considerazioni aggiuntive per i test di strumentazione Android](#)
- [Analisi dei test in modalità standard](#)

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.

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](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 vuoi 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 [Crea un progetto in AWS Device Farm](#).

4. Se il pulsante Create a new run (Crea una nuova sessione è visualizzato, selezionarlo.
5. Nella pagina Scegli applicazione, seleziona Scegli file.
6. Individua e seleziona il file dell'app Android. Il file deve essere di tipo .apk.

7. Seleziona Successivo.
8. Nella pagina Configura, nella sezione Setup test framework, scegli Strumentazione, quindi seleziona Scegli file.
9. Individuare e selezionare il file .apk che contiene i test.
10. Scegli Avanti, quindi completa le istruzioni rimanenti per selezionare i dispositivi e avviare l'esecuzione.

Acquisizione di 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`

Considerazioni aggiuntive per i test di strumentazione 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 [androidJUnitandroid.support.test.runner.A](#) 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
```

Utilizzo dei test iOS in AWS Device Farm

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

Framework di test delle applicazioni iOS

Sono disponibili i seguenti test per i dispositivi iOS.

- [Lavorare con Appium e AWS Device Farm](#)
- [Lavorare con XCTest per iOS e AWS Device Farm](#)
- [XCTestInterfaccia utente](#)

Tipi di test integrati per iOS

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

- [Integrato: fuzz \(Android e iOS\)](#)

Lavorare con XCTest per iOS e AWS Device Farm

Con Device Farm, puoi utilizzare il XCTest framework per testare la tua app su dispositivi reali. Per ulteriori informazioni in merito a 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.

Argomenti

- [Creazione dei pacchetti per l'XCTest esecuzione](#)
- [Caricamento dei pacchetti per la XCTest corsa su Device Farm](#)

Creazione dei pacchetti per l'XCTest esecuzione

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-iphonios, 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-iphonios 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.

Caricamento dei pacchetti per la 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](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 [Crea un progetto in AWS Device Farm](#).

Altrimenti, nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.

3. Scegliete il progetto che desiderate utilizzare per eseguire il test.
4. Scegliere Create a new run (Crea una nuova sessione).
5. Nella pagina Scegli l'applicazione, scegli App per dispositivi mobili.
6. Seleziona Scegli file.
7. Individuare il file `.ipa` per l'app e caricarlo.

Note

Il pacchetto `.ipa` deve essere compilato per il test.

8. Al termine del caricamento, scegli Avanti.
9. Nella pagina Configura, nella sezione Setup test framework, scegli XCTest. Quindi, seleziona Scegli file.
10. Cerca il `.zip` file che contiene il XCTest pacchetto per la tua app e caricalo.
11. Al termine del caricamento, scegli Avanti.
12. 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.
13. Dopo aver configurato la corsa, nella pagina Rivedi e avvia l'esecuzione, scegli Conferma e avvia l'esecuzione.

Device Farm esegue il test e mostra i risultati nella console.

Utilizzo del framework di test XCTest dell'interfaccia utente per iOS e AWS Device Farm

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

Argomenti

- [Cos'è il framework di test XCTest dell'interfaccia utente?](#)

- [Prepara i test XCTest dell'interfaccia utente iOS](#)
- [Carica i test XCTest dell'interfaccia utente iOS](#)
- [Acquisizione di schermate nei test XCTest dell'interfaccia utente iOS](#)

Cos'è il framework di test XCTest dell'interfaccia utente?

XCTestUI framework UI è il nuovo framework di test introdotto con Xcode 7. Questo framework si estende XCTest con le funzionalità di test dell'interfaccia utente. Per ulteriori informazioni, consulta la sezione [User Interface Testing](#) nella libreria per sviluppatori di iOS.

Prepara i test XCTest dell'interfaccia utente iOS

Il tuo pacchetto test runner XCTest dell'interfaccia utente iOS deve essere contenuto in un file.ipa correttamente formattato.

Per creare un file.ipa, posiziona il tuo pacchetto my-project-name UITest -Runner.app in una directory Payload vuota. Archiviare poi la directory Payload in un file .zip, quindi modificare l'estensione del file in .ipa. Il bundle* UITest -Runner.app è prodotto da Xcode quando crei il tuo progetto per i test. Si trova nella directory Products per il progetto.

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](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 vuoi 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 [Crea un progetto in AWS Device Farm](#)

4. Se il pulsante Create a new run (Crea una nuova sessione è visualizzato, selezionarlo.
5. Nella pagina Scegli applicazione, seleziona Scegli file.
6. 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.

7. Seleziona Successivo.
8. Nella pagina Configura, nella sezione Setup test framework, scegli XCTestUI, quindi seleziona Scegli file.
9. Cerca e scegli il file.ipa che contiene il tuo test runner XCTest dell'interfaccia utente iOS.
10. Scegli Avanti, quindi completa le istruzioni rimanenti per selezionare i dispositivi su cui eseguire i test e iniziare l'esecuzione.

Acquisizione di schermate nei test XCTest dell'interfaccia utente iOS

XCTestUI test dell'interfaccia utente acquisiscono automaticamente gli screenshot per ogni fase dei test. Queste schermate sono visualizzate nel rapporto di test di Device Farm. Non è richiesto codice supplementare.

Utilizzo dei test delle 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 "Appium"](#)

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.

Utilizzo dei test integrati in AWS Device Farm

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

Tipi di test integrati

I test integrati permettono di provare le app senza dover scrivere script.

- [Integrato: fuzz \(Android e iOS\)](#)

Utilizzo del fuzz test integrato per Device Farm

Device Farm offre un tipo di fuzz test integrato.

Cos'è il fuzz test integrato?

I test integrati Fuzz inviano casualmente eventi di interfaccia utente ai dispositivi, quindi riportano i risultati.

Usa il tipo di test fuzz integrato

Usa la console Device Farm per eseguire il fuzz test integrato.

1. Accedere alla console Device Farm all'[indirizzo https://console.aws.amazon.com/devicefarm](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 [Crea un progetto in AWS Device Farm](#).

4. Se il pulsante Create a new run (Crea una nuova sessione è visualizzato, selezionarlo.
5. Nella pagina Scegli applicazione, seleziona Scegli file.

6. Individuare e selezionare il file dell'app dove eseguire il test integrato fuzz.
7. Seleziona Successivo.
8. Nella pagina Configura, nella sezione Setup test framework, scegli Built-in: Fuzz.
9. 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.
10. Scegli Avanti, quindi completa le istruzioni rimanenti per selezionare i dispositivi e avviare l'esecuzione.

Utilizzo di 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 dei end-to-end test: il pacchetto di test non viene analizzato per rilevare tutti i test della suite, evitando il sovraccarico di preelaborazione/post-elaborazione.
- 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 in formato YAML (test spec). 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 CLI [Crea un'esecuzione di test in Device Farm](#)

Argomenti

- [Sintassi delle specifiche di test](#)

- [Esempio di specifiche di test](#)
- [Utilizzo dell'ambiente di test Amazon Linux 2 per test Android](#)
- [Variabili di ambiente](#)
- [Migrazione dei test da un ambiente di test standard a un ambiente di test personalizzato](#)
- [Estensione degli ambienti di test personalizzati in Device Farm](#)

Sintassi delle specifiche di test

Questa è la struttura del file di specifica di test YAML:

```
version: 0.1

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

artifacts:
  - location
  - location
```

La specifica di test contiene quanto segue:

version

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

phases

Questa sezione contiene gruppi di comandi eseguiti durante una sessione di test.

I nomi delle fasi di test ammessi sono:

install

Facoltativo.

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

Facoltativo.

I comandi, se presenti, eseguiti prima della tua sessione di test automatizzati.

test

Facoltativo.

I comandi eseguiti durante la tua sessione di test automatizzati. Se un comando nella fase di test ha esito negativo, il test viene contrassegnato come non riuscito.

post_test

Facoltativo.

I comandi, se presenti, eseguiti dopo la tua sessione di test automatizzati.

artifacts

Facoltativo.

Device Farm raccoglie elementi 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 [Utilizzo 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](#).

Esempio di specifiche di test

Questo è un esempio di una specifica di test YAML di Device Farm che configura un'esecuzione di test Appium Java TestNG:

```
version: 0.1

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

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

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

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

  # For iOS tests, you can use the Node.JS tools nvm, npm, and avm to setup your
  # environment. By
  # default, Node.js versions 16.20.2 and 14.19.3 are available on the test host.
```

```
install:
  commands:
    # The Appium server is written using Node.js. In order to run your desired
    version of Appium,
    # you first need to set up a Node.js environment that is compatible with your
    version of Appium.
    - |-
      if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "Android" ];
      then
        devicefarm-cli use node 16;
      else
        # For iOS, use "nvm use" to switch between the two preinstalled NodeJS
        versions 14 and 16,
        # and use "nvm install" to download a new version of your choice.
        nvm use 16;
      fi;
    - node --version

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

```
# For Appium version 2, for Android tests, Device Farm automatically updates the
preinstalled
# UIAutomator2 driver over time to incorporate the latest minor and patch
versions for its major
# version 2. If you want to install a specific version of the driver, you can use
the Appium
# extension CLI to uninstall the existing UIAutomator2 driver and install your
desired version:
# - |-
# if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "Android" ];
# then
#   appium driver uninstall uiautomator2;
#   appium driver install uiautomator2@2.34.0;
# fi;

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

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

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

```
# The pre-test phase contains commands for setting up your test environment.
pre_test:
  commands:
    # Device farm provides different pre-built versions of WebDriverAgent, an
    # essential Appium
    # dependency for iOS devices, and each version is suggested for different
    # versions of Appium:
    # DEVICEFARM_WDA_DERIVED_DATA_PATH_V8: this version is suggested for Appium 2
    # DEVICEFARM_WDA_DERIVED_DATA_PATH_V7: this version is suggested for Appium 1
    # Additionally, for iOS versions 16 and below, the device unique identifier
    # (UDID) needs
    # to be slightly modified for Appium tests.
    - |-
      if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "iOS" ];
      then
        if [ $(appium --version | cut -d "." -f1) -ge 2 ];
        then
          DEVICEFARM_WDA_DERIVED_DATA_PATH=$DEVICEFARM_WDA_DERIVED_DATA_PATH_V8;
        else
          DEVICEFARM_WDA_DERIVED_DATA_PATH=$DEVICEFARM_WDA_DERIVED_DATA_PATH_V7;
        fi;

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

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

    # We recommend starting the Appium server process in the background using the
    # command below.
    # The Appium server log will be written to the $DEVICEFARM_LOG_DIR directory.
```

```

# The environment variables passed as capabilities to the server will be
automatically assigned
# during your test run based on your test's specific device.
# For more information about which environment variables are set and how they're
set, please see
# https://docs.aws.amazon.com/devicefarm/latest/developerguide/custom-test-
environment-variables.html
- |-
if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "Android" ];
then
  appium --base-path=$APPIUM_BASE_PATH --log-timestamp \
    --log-no-colors --relaxed-security --default-capabilities \
    "{\"appium:deviceName\": \"\$DEVICEFARM_DEVICE_NAME\", \
    \"platformName\": \"\$DEVICEFARM_DEVICE_PLATFORM_NAME\", \
    \"appium:app\": \"\$DEVICEFARM_APP_PATH\", \
    \"appium:udid\": \"\$DEVICEFARM_DEVICE_UDID\", \
    \"appium:platformVersion\": \"\$DEVICEFARM_DEVICE_OS_VERSION\", \
    \"appium:chromedriverExecutableDir\":
\"\$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR\", \
    \"appium:automationName\": \"UiAutomator2\"}" \
    >> $DEVICEFARM_LOG_DIR/appium.log 2>&1 &
else
  appium --base-path=$APPIUM_BASE_PATH --log-timestamp \
    --log-no-colors --relaxed-security --default-capabilities \
    "{\"appium:deviceName\": \"\$DEVICEFARM_DEVICE_NAME\", \
    \"platformName\": \"\$DEVICEFARM_DEVICE_PLATFORM_NAME\", \
    \"appium:app\": \"\$DEVICEFARM_APP_PATH\", \
    \"appium:udid\": \"\$DEVICEFARM_DEVICE_UDID_FOR_APPIUM\", \
    \"appium:platformVersion\": \"\$DEVICEFARM_DEVICE_OS_VERSION\", \
    \"appium:derivedDataPath\": \"\$DEVICEFARM_WDA_DERIVED_DATA_PATH\", \
    \"appium:usePrebuiltWDA\": true, \
    \"appium:automationName\": \"XCUITest\"}" \
    >> $DEVICEFARM_LOG_DIR/appium.log 2>&1 &
fi;

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

```

```
    echo "Waiting for Appium to start on port 4723...";
    sleep 1;
done;

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

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

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

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

Utilizzo dell'ambiente di test Amazon Linux 2 per test Android

AWSDevice Farm utilizza macchine host Amazon Elastic Compute Cloud (EC2) che eseguono Amazon Linux 2 per eseguire test Android. Quando si pianifica 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 di test Amazon Linux 2 è l'ambiente di test Android più recente, che sostituisce il precedente sistema basato su Ubuntu. Utilizzando il file delle specifiche di test, puoi scegliere di eseguire i test Android sull'ambiente Amazon Linux 2.

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

- [Lingue e versioni di runtime supportate](#)
- [Intervalli IP supportati](#)
- [Lo strumento devicefarm-cli](#)
- [Selezione dell'host di test Android](#)
- [Esempio di file di specifiche di test](#)
- [Migrazione all'host di test Amazon Linux 2](#)

Lingue e versioni di runtime supportate

L'host di test Amazon Linux 2 è preinstallato con molte delle librerie software necessarie per supportare i framework di test di Device Farm, fornendo un ambiente di test pronto all'avvio. Per qualsiasi altro software richiesto, puoi modificare il file delle specifiche di test per installarlo dal pacchetto di test, scaricarlo da Internet o accedere a fonti private interne al tuo VPC ([VPCENI](#)) per

ulteriori informazioni, consulta la sezione dedicata). Per ulteriori informazioni, consulta l'esempio del [file Test spec](#).

Le seguenti versioni del software sono attualmente disponibili sull'host:

Libreria software	Versione del software	Comando da utilizzare nel file delle specifiche di test
Python	3.8	<code>devicefarm-cli use python 3.8</code>
	3.9	<code>devicefarm-cli use python 3.9</code>
	3.10	<code>devicefarm-cli use python 3.10</code>
	3,11	<code>devicefarm-cli use python 3.11</code>
Java	8	<code>devicefarm-cli use java 8</code>
	11	<code>devicefarm-cli use java 11</code>
	17	<code>devicefarm-cli use java 17</code>
NodeJS	16	<code>devicefarm-cli use node 16</code>
	18	<code>devicefarm-cli use node 18</code>
	20	<code>devicefarm-cli use node 20</code>
Ruby	2.7	<code>devicefarm-cli use ruby 2.7</code>

	3.2	<code>devicefarm-cli use ruby 3.2</code>
Appium	1	<code>devicefarm-cli use appium 1</code>
	2	<code>devicefarm-cli use appium 2</code>

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.

Intervalli IP supportati

I clienti spesso hanno bisogno di conoscere l'intervallo IP da cui proviene il traffico di Device Farm, in particolare per configurare i firewall e le impostazioni di sicurezza. Per gli host EC2 di test Amazon, l'intervallo IP comprende l'intera `us-west-2` regione. 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 NAT gateway, limitando l'intervallo IP ai seguenti indirizzi:

Intervalli IP

44.236.137,143

5213,151,44

5235,189,1991

54201.250,26

Lo strumento `devicefarm-cli`

L'host di test di Amazon Linux 2 utilizza uno strumento di gestione delle versioni standardizzato chiamato `devicefarm-cli` per selezionare le versioni del software. Questo strumento è separato dal Device Farm Test Host AWS CLI ed è disponibile solo sul Device Farm Test Host.

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.

Lo snippet riportato di seguito mostra la pagina di: `help 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 usare lo strumento per cambiare la versione di Python da **3.10** in **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 **1** in **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`.

Selezione dell'host di test Android

Warning

La versione precedente di Android Test Host non sarà più disponibile il 21 ottobre 2024. Tieni presente che il processo di deprecazione è suddiviso in diverse date:

- Il 22 aprile 2024, i lavori provenienti da ogni nuovo account verranno indirizzati all'host di test aggiornato.
- Il 2 settembre 2024, tutti i file delle specifiche di test nuovi o modificati devono essere indirizzati all'host di test aggiornato.
- Il 21 ottobre 2024, i lavori non potranno più essere eseguiti sull'host di test legacy.

Imposta i file delle specifiche di test sull'`amazon_linux_2`host per evitare problemi di compatibilità.

Per i test Android, Device Farm richiede il seguente campo nel file delle specifiche di test per scegliere l'host di test Amazon Linux 2:

```
android_test_host: amazon_linux_2 | legacy
```

`amazon_linux_2`Usalo per eseguire i test sull'host di test Amazon Linux 2:

```
android_test_host: amazon_linux_2
```

Scopri di più sui vantaggi di Amazon Linux 2 [qui](#).

Device Farm consiglia di utilizzare l'host Amazon Linux 2 per i test Android anziché l'ambiente host legacy. Se preferisci utilizzare l'ambiente legacy, usalo `legacy` per eseguire i test sull'host di test legacy:

```
android_test_host: legacy
```

Per impostazione predefinita, i file delle specifiche di test senza una selezione di host di test verranno eseguiti sull'host di test legacy.

Sintassi obsoleta

Di seguito è riportata la sintassi obsoleta per scegliere Amazon Linux 2 nel file delle specifiche di test:

```
preview_features:  
  android_amazon_linux_2_host: true
```

Se utilizzi questo flag, i tuoi test continueranno a essere eseguiti su Amazon Linux 2. Tuttavia, consigliamo vivamente di rimuovere la sezione `preview_features` flags e sostituirla con il nuovo `android_test_host` campo per evitare spese di manutenzione future.

Warning

L'utilizzo di entrambi `android_amazon_linux_2_host` i flag `android_test_host` e nel file delle specifiche di test restituirà un errore. Ne dovrebbe essere usato solo uno; lo consigliamo. `android_test_host`

Esempio di file di specifiche di test

Il seguente frammento è un esempio di file delle specifiche di test di Device Farm che configura un test Appium NodeJS eseguito utilizzando l'host di test Amazon Linux 2 per Android:

```
version: 0.1  
  
# This flag enables your test to run using Device Farm's Amazon Linux 2 test host. For  
# more information,  
# please see https://docs.aws.amazon.com/devicefarm/latest/developerguide/amazon-  
linux-2.html  
android_test_host: amazon_linux_2  
  
# Phases represent collections of commands that are executed during your test run on  
# the test host.  
phases:  
  
  # The install phase contains commands for installing dependencies to run your tests.  
  # For your convenience, certain dependencies are preinstalled on the test host. To  
  # lean about which  
  # software is included with the host, and how to install additional software, please  
  # see:
```

```
# https://docs.aws.amazon.com/devicefarm/latest/developerguide/amazon-linux-2-
supported-software.html

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

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

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

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

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

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

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

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

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

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

```

    \"appium:platformVersion\": \"${DEVICEFARM_DEVICE_OS_VERSION}\", \
    \"appium:chromedriverExecutableDir\":
\"$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR\", \
    \"appium:automationName\": \"UiAutomator2\"} \
    >> $DEVICEFARM_LOG_DIR/appium.log 2>&1 &

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

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

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

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

# Artifacts are a list of paths on the filesystem where you can store test output and
reports.

```

```
# All files in these paths will be collected by Device Farm.
# These files will be available through the ListArtifacts API as your "Customer
  Artifacts".
artifacts:
  # By default, Device Farm will collect your artifacts from the $DEVICEFARM_LOG_DIR
  directory.
  - $DEVICEFARM_LOG_DIR
```

Migrazione all'host di test Amazon Linux 2

Warning

La versione precedente di Android Test Host non sarà più disponibile il 21 ottobre 2024. Tieni presente che il processo di deprecazione è suddiviso in diverse date:

- Il 22 aprile 2024, i lavori provenienti da ogni nuovo account verranno indirizzati all'host di test aggiornato.
- Il 2 settembre 2024, tutti i file delle specifiche di test nuovi o modificati devono essere indirizzati all'host di test aggiornato.
- Il 21 ottobre 2024, i lavori non potranno più essere eseguiti sull'host di test legacy.

Imposta i file delle specifiche di test sull'`amazon_linux_2host` per evitare problemi di compatibilità.

Per migrare i test esistenti dall'host legacy al nuovo host Amazon Linux 2, sviluppa nuovi file di specifiche di test basati su quelli preesistenti. L'approccio consigliato consiste nell'iniziare con i nuovi file di specifiche di test predefiniti per i tipi di test. Quindi, migra i comandi pertinenti dal vecchio file delle specifiche di test a quello nuovo, salvando il vecchio file come backup. Ciò consente di sfruttare le specifiche predefinite ottimizzate per il nuovo host riutilizzando al contempo il codice esistente. Ti assicura di ottenere tutti i vantaggi del nuovo host configurato in modo ottimale per i tuoi test, mantenendo al contempo le specifiche di test precedenti come riferimento durante l'adattamento dei comandi al nuovo ambiente.

I seguenti passaggi possono essere utilizzati per creare un nuovo file di specifiche di test di Amazon Linux 2 riutilizzando i comandi dal vecchio file di specifiche di test:

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

2. Vai al progetto Device Farm contenente i tuoi test di automazione.
3. Scegli Crea una nuova esecuzione di test nel progetto.
4. Scegli un'app e un pacchetto di test utilizzati in precedenza per il tuo framework di test.
5. Scegli Esegui il test in un ambiente personalizzato.
6. Scegli il file delle specifiche di test che stai attualmente utilizzando per i test sull'host di test legacy dal menu a discesa delle specifiche di test.
7. Copia il contenuto di questo file e incollalo localmente in un editor di testo per consultarli in seguito.
8. Nel menu a discesa delle specifiche di test, modifica la selezione delle specifiche di test con il file di specifiche di test predefinito più recente.
9. Scegli Modifica e accederai all'interfaccia di modifica delle specifiche di test. Noterai che, nelle prime righe del file delle specifiche di test, ha già scelto il nuovo host di test:

```
android_test_host: amazon_linux_2
```

10. [Consulta la sintassi per la selezione degli host di test qui e le principali differenze tra gli host di test qui.](#)
11. Aggiungi e modifica selettivamente i comandi dal file delle specifiche di test salvato localmente dal passaggio 6 nel nuovo file di specifiche di test predefinito. Quindi, scegli Salva con nome per salvare il nuovo file delle specifiche. Ora puoi pianificare le esecuzioni di test sull'host di test Amazon Linux 2.

Differenze tra gli host di test nuovi e quelli precedenti

Quando modifichi il file delle specifiche di test per utilizzare l'host di test Amazon Linux 2 e trasferisci i test dall'host di test precedente, tieni presente queste differenze chiave nell'ambiente:

- Selezione delle versioni del software: in molti casi, le versioni software predefinite sono cambiate, quindi se prima non hai selezionato esplicitamente la tua versione del software nell'host di test Legacy, potresti volerla specificare ora nell'host di test Amazon Linux 2 utilizzando [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 `envm`, `pyenvvm`, 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](#).
- Tutti i percorsi di file codificati nel file delle specifiche di test dell'host Legacy come percorsi assoluti molto probabilmente non funzioneranno come previsto nell'host di test di Amazon Linux 2; in genere 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 dei file di test spec. Inoltre, tieni presente che la maggior parte dei file binari necessari per il test si trova nell'`PATH`host, in modo che siano immediatamente eseguibili dal file spec usando solo il loro nome (come `appium`).
- Al momento la raccolta dei dati sulle prestazioni non è supportata sul nuovo host di test.
- Versione del sistema operativo: l'host di test legacy era basato sul sistema operativo Ubuntu, mentre quello nuovo è basato su Amazon Linux 2. Di conseguenza, gli utenti potrebbero notare alcune differenze nelle librerie di sistema disponibili e nelle versioni delle librerie di sistema.
- Per gli utenti di Appium Java, il nuovo host di test non contiene alcun JAR file preinstallato nel percorso di classe, mentre l'host precedente ne conteneva uno per il framework TestNG (tramite una variabile di ambiente). `$DEVICEFARM_TESTNG_JAR` Consigliamo ai clienti di impacchettare JAR i file necessari per i propri framework di test all'interno del pacchetto di test e di rimuovere le istanze della variabile dai file delle `$DEVICEFARM_TESTNG_JAR` specifiche di test. Per ulteriori informazioni, consulta [Working with Appium and AWS Device Farm](#).
- Per gli utenti di Appium, la variabile di `$DEVICEFARM_CHROMEDRIVER_EXECUTABLE` ambiente è stata rimossa a favore di un nuovo approccio per consentire ai clienti di accedere a Chromedriver per Android. Vedi il nostro [file di specifiche di test predefinito](#) per un esempio, che utilizza una nuova variabile di ambiente. `$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR`

Note

Consigliamo vivamente di mantenere il comando del server Appium esistente dal file di specifiche di test predefinito così com'è.

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.

Variabili di ambiente

Le variabili di ambiente rappresentano valori utilizzati dai tuoi test automatizzati. Puoi utilizzare queste variabili di ambiente nei tuoi file YAML e nel codice di test. In un ambiente di test personalizzato, Device Farm popola dinamicamente le variabili di ambiente in fase di esecuzione.

Argomenti

- [Variabili di ambiente comuni](#)
- [Variabili di ambiente Appium Java JUnit](#)
- [Variabili di ambiente Appium Java TestNg](#)
- [Variabili di ambiente XCUITest](#)

Variabili di ambiente comuni

Test Android

Questa sezione descrive le variabili di ambiente personalizzate comuni ai test della piattaforma Android supportati da Device Farm.

\$DEVICEFARM_DEVICE_NAME

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

\$DEVICEFARM_DEVICE_PLATFORM_NAME

Il nome della piattaforma del dispositivo. È Android o iOS.

\$DEVICEFARM_DEVICE_OS_VERSION

La versione del sistema operativo del dispositivo.

\$DEVICEFARM_APP_PATH

Il percorso all'app mobile sulla macchina host dove vengono eseguiti i test. Il percorso dell'app è disponibile solo per app mobili.

\$DEVICEFARM_DEVICE_UDID

L'identificatore univoco del dispositivo mobile che esegue il test automatizzato.

\$DEVICEFARM_LOG_DIR

Il percorso ai file di log generati durante la sessione di test. Per impostazione predefinita, tutti 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_CHROMEDRIVER_EXECUTABLE_DIR

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

\$ANDROID_HOME

Il percorso della directory di installazione di Android SDK.

Note

La variabile di ANDROID_HOME ambiente è disponibile solo sull'host di test Amazon Linux 2 per Android.

Test iOS

Questa sezione descrive le variabili di ambiente personalizzate comuni ai test della piattaforma iOS supportati da Device Farm.

\$DEVICEFARM_DEVICE_NAME

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

\$DEVICEFARM_DEVICE_PLATFORM_NAME

Il nome della piattaforma del dispositivo. È Android o iOS.

\$DEVICEFARM_APP_PATH

Il percorso all'app mobile sulla macchina host dove vengono eseguiti i test. Il percorso dell'app è disponibile solo per app mobili.

\$DEVICEFARM_DEVICE_UDID

L'identificatore univoco del dispositivo mobile che esegue il test automatizzato.

\$DEVICEFARM_LOG_DIR

Il percorso ai file di log generati durante la sessione di test.

\$DEVICEFARM_SCREENSHOT_PATH

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

Variabili di ambiente Appium Java JUnit

Questa sezione descrive le variabili di ambiente utilizzate dai test Appium Java JUnit in un ambiente di test personalizzato.

\$DEVICEFARM_TESTNG_JAR

Il percorso al file .jar TestNG.

\$DEVICEFARM_TEST_PACKAGE_PATH

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

Variabili di ambiente Appium Java TestNg

Questa sezione descrive le variabili di ambiente utilizzate dai test Appium Java TestNG in un ambiente di test personalizzato.

\$DEVICEFARM_TESTNG_JAR

Il percorso al file .jar TestNG.

\$DEVICEFARM_TEST_PACKAGE_PATH

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

Variabili di ambiente XCUITest

\$DEVICEFARM_XCUITESTRUN_FILE

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.

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

La seguente guida spiega come passare da una modalità di esecuzione dei test standard a una modalità di esecuzione personalizzata. 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.

Considerazioni sulla 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 sarebbero 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.

4. **Obsolescenza:** i seguenti tipi di test saranno dichiarati obsoleti entro la fine di dicembre 2023 nella modalità di esecuzione standard:

- Appium (tutte le lingue)
- Calabash
- XCTest
- UI Automation
- UI Automator
- Test Web
- Explorer integrato

Una volta obsoleti, non sarà possibile utilizzare questi framework in modalità standard. È invece possibile utilizzare la modalità personalizzata per i tipi di test elencati sopra.

i Tip

Se il tuo caso d'uso coinvolge 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. Accedere AWS Management Console e aprire la console Device Farm all'[indirizzo https://console.aws.amazon.com/devicefarm/](https://console.aws.amazon.com/devicefarm/).
2. Scegli il tuo progetto e poi avvia una nuova esecuzione di automazione.
3. Carica la tua app (o web app selezionata), 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'ScheduleRunAPI.](#)

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.

XCUITest per iOS

Non devi apportare modifiche per spostare i tuoi test XCUITest per iOS in un ambiente di test personalizzato.

Estensione degli ambienti di test personalizzati in Device Farm

La modalità personalizzata di 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.

Impostazione di un PIN

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

- Il dispositivo deve avere Android 8 o versioni successive.
- PINDeve essere rimosso dopo il completamento del test.

Per impostare i PIN test, utilizzate le `post_test` fasi `pre_test` e per impostarli e rimuoverliPIN, 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"
```

Quando inizia la suite di test, viene impostato il PIN 1234. Dopo l'uscita della suite di test, PIN viene rimossa.

Warning

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

Accelerazione dei test basati su Appium grazie alle funzionalità desiderate

Quando si utilizza Appium, è possibile che la suite di test in modalità standard sia 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 tra virgolette e derivata da shell escape. JSON

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 PIN codice, il che velocizza i test dopo un evento di screen off 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 HTTP richieste al momento specificato. `webdriverAgentUrl` Se `webdriverAgent` non è già attivo e funzionante, Appium può impiegare del tempo all'inizio di una suite di test per avviare la `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 dispositivo `8100`, quindi inoltrarlo alla porta locale dell'host di test `8100` (ciò consente `webdriverAgentUrl` di impostare il valore su `http://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](#).

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

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/XXXXXXXXXXXXXXXXXXXXXXXXXXXX

```

Per ulteriori informazioni sull'uso dei webhook con Slack, consulta [Invio del primo messaggio Slack utilizzando Webhook nel riferimento a Slack](#). API

Non sei limitato a usare per chiamare i webhook. curl 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 API delle chiavi 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 credenziali IAM temporanee, generate al di fuori della 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, vedere [Richiesta di credenziali di sicurezza temporanee nella Guida per l'utente](#). IAM

Aggiungere file aggiuntivi al pacchetto di test

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 (ZIPIPA,APK,JAR, ecc.) sono formati di archivio dei pacchetti che supportano operazioni standardZIP.

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 IPA dei file. Per quanto riguarda IPA i file, specialmente se utilizzati conXCUI Tests, ti consigliamo di mettere tutti i file aggiuntivi in una posizione leggermente diversa a causa della progettazione AWS Device Farm dei pacchetti 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, aggiungi eventuali file aggiuntivi alla directory che termina con *.app* all'interno del *Payload* elenco. Ad esempio, i seguenti comandi 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 ZIP file, AWS Device Farm decomprimerà automaticamente il file caricato prima del test e lascerà i file decompressi nella posizione in cui si trova *\$DEVICEFARM_TEST_PACKAGE_PATH* variabile d'ambiente. (Ciò significa che se hai aggiunto un file chiamato *extra_file* alla radice dell'archivio, come nel primo esempio, 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 con il tuo 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 personalizzato 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 lo è ZIP (ad esempio APKIPA,, o JAR file), il file del pacchetto caricato stesso si trova in `$DEVICEFARM_TEST_PACKAGE_PATH`. Poiché si tratta ancora di file in formato di archivio, è possibile decomprimere il file per accedere ai file aggiuntivi dall'interno. Ad esempio, il comando seguente decomprimerà il contenuto del pacchetto di test (for APKIPA, o JAR files) in `/tmp` cartella:

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

Nel caso di un JAR file APK or, troverai i tuoi file aggiuntivi decompressi in `/tmp` directory (ad es. `/tmp/extra_file`). Nel caso di un IPA file, 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 di `Payload` elenco. Ad esempio, in base all'IPAesempio precedente, il file verrebbe trovato nella posizione `/tmp/Payload/ADFiOSReferenceAppUITests-Runner.app/extra_file` (referenziabile come `/tmp/Payload/*.app/extra_file`).

Utilizzo dell'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 reale 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.

Argomenti

- [Crea una sessione di accesso remoto in AWS Device Farm](#)
- [Usa una sessione di accesso remoto in AWS Device Farm](#)
- [Ottieni i risultati di una sessione di accesso remoto in AWS Device Farm](#)

Crea una sessione di accesso remoto in AWS Device Farm

Per ulteriori informazioni sulle sessioni di accesso remoto, consulta [Sessioni](#).

- [Prerequisiti](#)
- [Crea un'esecuzione di test \(console\)](#)

- [Fasi successive](#)

Prerequisiti

- Crea un progetto in Device Farm. Segui le istruzioni riportate in [Crea un progetto in AWS Device Farm](#), poi torna a questa pagina.

Crea una sessione con la console Device Farm

1. Accedi alla console Device Farm all'indirizzo <https://console.aws.amazon.com/devicefarm>.
2. Nel pannello di navigazione di Device Farm, scegli **Test dei dispositivi mobili**, quindi **Progetti**.
3. Se hai già un progetto, selezionalo dall'elenco. Altrimenti, crea un progetto seguendo le istruzioni in [Crea un progetto in AWS Device Farm](#).
4. Nella scheda **Remote access (Accesso remoto)**, selezionare **Start a new session (Avvia una nuova sessione)**.
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. È possibile effettuare la ricerca per:
 - Nome
 - Piattaforma
 - Fattore di forma
 - Tipo di flotta
6. In **Session name (Nome sessione)**, immettere un nome per la sessione.
7. Selezionare **Confirm and start session (Conferma e avvia sessione)**.

Fasi successive

Device Farm avvia la sessione non appena il dispositivo richiesto è disponibile, in genere entro pochi minuti. La richiesta viene visualizzata una finestra di dialogo fino all'inizio della sessione. Per annullare la richiesta di sessione, selezionare **Cancel request (Annulla richiesta)**.

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'avvio della sessione, è possibile interagire con il dispositivo nel browser Web.

Usa 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](#)
- [Usa una sessione nella console Device Farm](#)
- [Fasi successive](#)
- [Suggerimenti e trucchi](#)

Prerequisiti

- Crea una sessione. Segui le istruzioni riportate in [Crea una sessione](#), poi torna a questa pagina.

Usa 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 vicino al nome del dispositivo.

Installazione di un'applicazione

Per installare un'applicazione sul dispositivo di sessione, in Installare le 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

AWS Device Farm non visualizza una conferma dopo l'installazione di un'app. Prova a interagire con l'icona dell'app per verificare se l'app è pronta per l'uso.

Quando carichi un'app, a volte potrebbe essere disponibile con un po' di ritardo. Controlla la barra delle applicazioni per vedere se l'app è disponibile.

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 la interrompi manualmente o non viene raggiunto il limite di 150 minuti. Per terminare la sessione, scegli [Interrompi sessione](#). Una volta interrotta la sessione puoi accedere al video acquisito e ai log generati. Per ulteriori informazioni, consulta [Ottieni i risultati della sessione](#).

Suggerimenti e trucchi

In alcune regioni AWS potrebbero verificarsi problemi a livello 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.

Ottieni i 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 [Usa 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 **Test** dei dispositivi mobili, quindi scegli **Progetti**.
2. Scegli 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.

Utilizzo di 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. 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 [Gestione dei dispositivi privati 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 sul tuo computer VPC che desideri testare su dispositivi mobili. Per ulteriori informazioni, consulta [Utilizzo dei servizi VPC endpoint Amazon 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

- [Gestione dei dispositivi privati in AWS Device Farm](#)
- [Selezione di dispositivi privati in un pool di dispositivi](#)
- [Ignorare la nuova firma dell'app sui dispositivi privati in Device Farm AWS](#)
- [Collaborazione con Amazon VPC in tutte AWS le regioni](#)
- [Disattivazione dei dispositivi privati](#)

Gestione dei 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 degli AWS Stati Uniti occidentali (Oregon) (us-west-2).

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.

Questo argomento spiega come creare un'istanza profilo ed eseguire altre attività comuni di gestione del dispositivo.

Argomenti

- [Creazione di un profilo dell'istanza](#)
- [Gestione di un'istanza privata del dispositivo](#)
- [Creare un'esecuzione di test o avviare una sessione di accesso remoto](#)
- [Passaggi successivi](#)

Creazione di un profilo dell'istanza

Per controllare il comportamento dei dispositivi privati durante un'esecuzione di test o una sessione di accesso remoto, è possibile creare o modificare un profilo di istanza in Device Farm. Non è necessario un profilo delle istanze per iniziare a utilizzare i dispositivi privati.

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.

Create a new instance profile ✕

Name
Name of the profile that can be attached to one or more private devices.

Description - optional
Description of the profile that can be attached to one or more private devices.

Reboot
If checked, the private device will reboot after use.

Reboot after use

Package cleanup
If checked, the packages installed during run time on the private device will be removed after use.

Package cleanup after use

Exclude packages from cleanup
Add fully qualified names of packages that you want to be excluded from cleanup after use. Example: com.test.example.

[+ Add new](#)

Cancel Save

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. Selezionare Salva.

Gestione di un'istanza privata del dispositivo

Se si dispone già di uno o più dispositivi privati nel parco istanze, è possibile visualizzare informazioni su e gestire determinate impostazioni per ogni istanza del dispositivo. È anche possibile richiedere un'ulteriore istanza del dispositivo privato.

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. (Facoltativo) Per richiedere una nuova istanza di dispositivo privato, scegli Richiedi istanza 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.

Edit device instances ✕

Instance ID
ID for the private device instance.

Mobile
Model of the private device.

Platform
Platform of the private device.

OS Version
OS version of the private device.

Status
Status of the private device.

Profile
Choose a profile to attach to the device.

Instance profile details

Name:

Reboot after use: false

Package Cleanup: false

Excluded Packages:

Labels
Labels are custom strings that can be attached to private devices.

 ✕

+ Add new

Cancel Save

- (Facoltativo) Per Profile (Profilo), scegliere un'istanza del profilo da collegare all'istanza del dispositivo. Questo può essere utile se si desidera escludere sempre un pacchetto app specifico, da attività di pulizia, ad esempio.
- (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.
- Selezionare Salva.

Creare un'esecuzione di test o avviare una sessione di accesso remoto

Dopo aver configurato un parco istanze di dispositivi privati, è possibile sessioni di test o avviare sessioni di accesso remoto con uno o più dispositivi privati.

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 di queste 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).

Create a new remote session

Choose a device

Select a device for an interactive session. Interested in unlimited, unmetered testing? [Purchase device slots](#)

Private device instances only

Show available devices only
(Note: When a device is 'AVAILABLE', your session will start in under a minute)

Find by name, platform, OS, form factor, or fleetType

	Name	Status	Platform	OS	Form factor	Instance Id	Labels
<input type="radio"/>	OnePlus 8T	AVAILABLE	Android	11	Phone	-	-
<input type="radio"/>	Samsung Galaxy Tab S7	AVAILABLE	Android	11	Tablet	-	-

Passaggi successivi

Dopo aver configurato i tuoi dispositivi privati, puoi anche gestire i tuoi dispositivi privati nei seguenti modi:

- [Ignorare la nuova firma dell'app su dispositivi privati](#)
- [Usa i servizi endpoint Amazon Virtual Private Cloud con Device Farm](#)

Per eliminare un profilo di istanza, nel menu Profili di istanza, scegli l'opzione di attivazione/disattivazione accanto all'istanza che desideri eliminare, quindi scegli Elimina.

Selezione di dispositivi privati in un pool di dispositivi

Per utilizzare dispositivi privati durante il test, puoi creare un pool di dispositivi che selezioni 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 sul dispositivo ARN
2. Regole basate sull'etichetta dell'istanza del dispositivo
3. Regole basate sull'istanza del dispositivo ARN

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, AWS Command Line Interface (AWS CLI) o Device Farm API per creare o modificare un pool di dispositivi con dispositivi privati utilizzando queste regole.

Argomenti

- [Dispositivo ARN](#)
- [Etichette delle istanze del dispositivo](#)
- [Istanza ARN](#)
- [Creazione di un pool di dispositivi privato con dispositivi privati \(console\)](#)
- [Creazione di un pool di dispositivi privati con dispositivi privati \(AWS CLI\)](#)
- [Creazione di un pool di dispositivi privato con dispositivi privati \(API\)](#)

Dispositivo ARN

Un dispositivo ARN è un identificatore che rappresenta un tipo di dispositivo anziché una specifica istanza di dispositivo fisico. Un tipo di dispositivo è definito dai seguenti attributi:

- L'ID della flotta del dispositivo
- Il dispositivo è OEM
- 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 ne avessi tre *Apple iPhone 13* dispositivi in versione *iOS 16.1.0* nella tua flotta privata, ogni dispositivo condividerebbe lo stesso dispositivoARN. Se dal tuo parco dispositivi venissero aggiunti o rimossi gli stessi attributi, il dispositivo ARN continuerebbe a rappresentare tutti i dispositivi disponibili nel tuo parco dispositivi per quel tipo di dispositivo.

Il dispositivo ARN 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 ARN regola del dispositivo garantisce che il pool di dispositivi possa continuare a selezionare i dispositivi in caso di guasto hardware.

Quando utilizzi una ARN regola 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 da quel dispositivoARN. 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 di quel dispositivo ARN 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 il dispositivo ARN 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 selezionare 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 disponibile.

Istanza ARN

Un'istanza di dispositivo ARN è un identificatore che rappresenta un'istanza fisica di dispositivo bare metal distribuita in un parco veicoli privato. Ad esempio, se ne avessi tre *iPhone 13* dispositivi su sistema operativo *15.0.0* nella vostra flotta privata, sebbene ogni dispositivo condivida lo stesso dispositivo ARN, ogni dispositivo avrebbe anche una propria istanza che ARN rappresenta solo quell'istanza.

L'istanza del dispositivo ARN è il modo meno affidabile per selezionare i dispositivi privati per un pool di dispositivi ed è consigliata solo se le etichette delle istanze del dispositivo ARNs e del dispositivo non si adattano al tuo caso d'uso. ARNsLe 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 sostitutivo avrà un'istanza di dispositivo diversa da ARN quella del dispositivo sostituito. Pertanto, se ti affidi all'istanza di dispositivo ARNs per il tuo pool di dispositivi, dovrai modificare manualmente la definizione delle regole del pool di dispositivi dall'utilizzo della vecchia ARN all'utilizzo della nuova 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 ARN regola di istanza del 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 i 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 CLI o 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. Scegliere Automated tests (Test automatici) e Create a new run (Crea una nuova sessione). La procedura guidata consente di scegliere la propria applicazione e configurare il test che si desidera eseguire.

5. Per il passaggio Seleziona dispositivi, scegli Crea pool di dispositivi e inserisci un nome e una descrizione opzionale per il tuo pool di dispositivi.
 - a. Per utilizzare ARN le regole 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 dispositivi private, perché questa opzione fa sì che il pool di dispositivi venga creato con ARN le regole delle istanze dei dispositivi (anziché con le ARN regole dei dispositivi).

The screenshot shows the 'Create device pool' dialog box. The 'Name' field contains 'MyPrivateDevicePool'. The 'Description - optional' field is empty. The 'Device selection method' section is highlighted with a red box and contains two radio buttons: 'Create dynamic device pool' (unselected) and 'Create static device pool' (selected). Below this, there is a checkbox for 'See private device instances only' which is unchecked. The 'Mobile devices (0/92)' section shows a search bar and a table with columns: Name, Status, Platform, OS, Form factor, Instance Id, and Labels. The table contains one row with a device icon, the name 'Available', status 'Available', platform 'Android', OS '10', form factor 'Phone', and an instance ID. 'Cancel' and 'Create' buttons are at the bottom right.

- 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'Operatore etichetta dell'istanza del dispositivo desiderata come Value.

The screenshot shows the 'Create device pool' dialog box. The 'Name' field contains 'MyPrivateDevicePool'. The 'Description - optional' field is empty. The 'Device selection method' section is highlighted with a red box and contains two radio buttons: 'Create dynamic device pool' (selected) and 'Create static device pool' (unselected). Below this, the 'Filter by device attribute' section is highlighted with a red box and contains a table with columns: Field, Operator, and Value. The table contains one row with 'Instance Labels' in the Field column, 'CONTAINS' in the Operator column, and 'Example' in the Value column. There is an 'Add a rule' button below the table. The 'Max devices' section has a text input field. A blue information box states: 'If you do not enter the max devices, we will pick all devices in our fleet that match the above rules'. The 'Mobile devices (0/92)' section shows a search bar and a table with columns: Name, Status, Platform, OS, Form factor, Instance Id, and Labels. The table is empty. 'Cancel' and 'Create' buttons are at the bottom right.

- c. Per utilizzare ARN le regole 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 AWS account.

Create device pool

Name
MyPrivateDevicePool

Description - optional
Enter a short description for your device pool

Device selection method
Use Rules to create a dynamic device pool that adapts as new devices become available (recommended) OR select devices individually to create a static device pool

Create dynamic device pool Create static device pool

See private device instances only

Mobile devices (0/92)

Find devices by attribute

Name	Status	Platform	OS	Form factor	Instance id	Labels
ⓘ [redacted]	Available	Android	10	Phone	[redacted]	-

Cancel Create

6. Scegli Crea.

Creazione di un pool di dispositivi privati con dispositivi privati (AWS CLI)

- Esegui il comando [create-device-pool](#).

Per informazioni sull'utilizzo di Device Farm con AWS CLI, vedere [Riferimento AWS CLI](#).

Creazione di un pool di dispositivi privato con dispositivi privati (API)

- Chiama il [CreateDevicePool](#) API.

Per informazioni sull'utilizzo di Device Farm API, vedere [Automazione di Device Farm](#).

Ignorare la nuova firma dell'app sui dispositivi privati in Device Farm AWS

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 incorporato con un profilo wildcard e rassegniamo le dimissioni dell'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 rinuncia all'app iOS comporta la rimozione di determinati diritti. Ciò include il gruppo di app, i domini associati, Game Center, HealthKit, la configurazione degli accessori wireless HomeKit, l'acquisto in-app, l'audio tra le app, Apple Pay, le notifiche push e la configurazione e il controllo. VPN

Su Android, abbandoniamo l'app. Ciò potrebbe interrompere funzionalità che dipendono dalla firma dell'app, come Google Maps per AndroidAPI. Può anche 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 il passaggio in cui AWS Device Farm firma nuovamente 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 [AWSDevice Farm FAQs](#) o [App](#).

Per ignorare la nuova firma dell'app per una sessione di test, seleziona Skip app re-signing (Ignora nuova firma dell'app) nella pagina Configure (Configura) quando crei la sessione di test.

Configure

Setup test framework

Select the test type you would like to use. If you do not have any scripts, select 'Built-in: Fuzz' or 'Built-in: Explorer' and we will fuzz test or explore your app

Built-in: Fuzz

No tests? No problem. We'll fuzz test your app by sending random events to it with no scripts required.

Event count

The number of events between 1 and 10000 that the UI Fuzz test should perform.

6000

Event throttle

The time in ms between 0 and 1000 that the UI fuzz test should wait between events.

50

Randomizer seed

A seed to use for randomizing the UI fuzz test. Using the same seed value between tests ensures identical event sequences.

Enter a randomizer seed

▼ Advanced Configuration (optional)

Configuration specific to Private Devices

App re-signing

If checked, this skips app re-signing and enables you to test with your own provisioning profile

Skip app re-signing

Other Configuration

Change default selection for enabling video and data capture - default "on"

Video recording

If checked, enables video recording during test execution.

Enable video recording

Note

Se utilizzi il XCTest framework, l'opzione Salta nuovamente la firma dell'app non è disponibile. Per ulteriori informazioni, consulta [Lavorare con XCTest per iOS e AWS Device Farm](#).

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 UDID il dispositivo, utilizzare il `ListDeviceInstances` funzionamento della Device FarmAPI.
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 questo scopo, puoi installare l'app da testare sul dispositivo privato o installare un'app fittizia firmata con lo stesso certificato dell'app che desideri testare. C'è un vantaggio nell'installazione dell'app fittizia registrata con lo stesso certificato. Dopo aver confermato l'attendibilità del profilo di configurazione o dello sviluppatore di app enterprise, tutte le app di quello sviluppatore sono ritenute attendibili sul dispositivo privato fino a quando non le elimini. Pertanto, quando carichi una nuova versione dell'app da testare, non dovrai confermare nuovamente l'attendibilità dello sviluppatore dell'app. 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.

Prima di iniziare la sessione di accesso remoto, segui i passaggi indicati [Creazione di un profilo dell'istanza](#) per creare o modificare un profilo di istanza in Device Farm. Nel profilo dell'istanza, aggiungi l'ID del pacchetto dell'app di test o dell'app fittizia 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 fittizia per il dispositivo utilizzando una sessione di accesso remoto, che ti consente di avviare l'app e di confermare l'attendibilità dello sviluppatore.

1. Seguire le istruzioni in [Crea 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).

Choose a device

Select a device for an interactive session.

Use my 1 unmetered iOS device slot ⓘ

Skip app re-signing ⓘ

Private device instances only

⚠ 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 fittizia o l'app che desideri testare all'impostazione **Exclude packages from cleanup** 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. Segui le istruzioni per confermare l'attendibilità del certificato dello sviluppatore.

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.

Collaborazione con Amazon VPC in tutte AWS le regioni

I servizi Device Farm sono disponibili solo nella regione Stati Uniti occidentali (Oregon) (us-west-2). Puoi usare Amazon Virtual Private Cloud (AmazonVPC) 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, vedi [Utilizzo dei servizi VPC endpoint Amazon 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 che non lo è us-west-2, puoi utilizzare VPC Peering per peerizzare quella regione VPC a un'altra 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 VPC di peering](#) nella Amazon VPC Peering Guide.

VPCpeering

È possibile eseguire il peering tra due VPCs aree geografiche diverse, purché abbiano blocchi distinti e non CIDR 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 CIDR notazione, vedere [RFC4632](#).

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 di questo esempio (denominato VPC-2) si trova in un'altra regione.

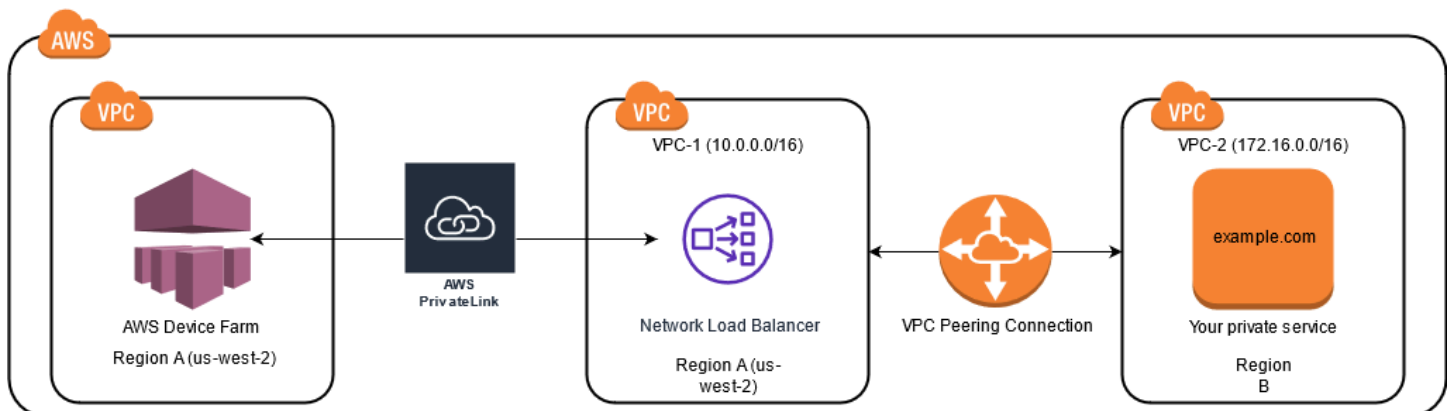
Esempio VPC interregionale di Device Farm

VPC Componente	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](#)
- [Passaggio 1: configura una connessione peering tra -1 e -2 VPC VPC](#)
- [Passaggio 2: aggiorna le tabelle delle rotte in VPC -1 e -2 VPC](#)

- [Fase 3: Creare un gruppo target](#)
- [Fase 4: Creare un Network Load Balancer](#)
- [Fase 5: Creare un servizio VPC endpoint](#)
- [Fase 6: Creare una configurazione VPC degli endpoint in Device Farm](#)
- [Fase 7: Creare un test](#)
- [Crea una rete scalabile con Transit Gateway](#)

Prerequisiti

Questo esempio richiede quanto segue:

- Due VPCs configurati con sottoreti contenenti blocchi non CIDR sovrapposti.
- VPC-1 deve trovarsi nella us-west-2 regione e contenere sottoreti per le zone di disponibilità e. us-west-2a 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 Peering Guide. VPC

Passaggio 1: configura una connessione peering tra -1 e -2 VPC VPC

Stabilisci una connessione di peering tra i due blocchi VPCs contenenti non CIDR sovrapposizioni. A tale scopo, consulta [Creare e accettare connessioni VPC peering](#) 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 connessione peering:

Nome

Device-Farm-Peering-Connection-1

VPCID (richiedente)

vpc-0987654321gfedcba (VPC-2)

Account

My account

Region

US West (Oregon) (us-west-2)

VPCID (Accettatore)

vpc-1234567890abcdefg (VPC-1)

Note

Assicurati di consultare le quote di connessione VPC peering quando stabilisci nuove connessioni peering. Per ulteriori informazioni, consulta le [VPCquote Amazon](#) nella Amazon VPC Peering Guide.

Passaggio 2: aggiorna le tabelle delle rotte in VPC -1 e -2 VPC

Dopo aver configurato una connessione peering, è necessario stabilire un percorso di destinazione tra i due VPCs per trasferire i dati tra di loro. Per stabilire questa rotta, 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 VPC peering](#) 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 dei VPC percorsi 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: Creare un gruppo target

Dopo aver impostato i percorsi di destinazione, puoi 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 fare in modo che quella risorsa statica indirizzi le richieste verso la tua destinazione reale.

Note

- Se hai come target una o più istanze Amazon Elastic Compute Cloud (Amazon) autonome, apri la console EC2 EC2 Amazon <https://console.aws.amazon.com/ec2/> all'indirizzo, quindi scegli Istanze.
- Se hai come target un gruppo Amazon EC2 Auto Scaling di istanze EC2 Amazon, devi associare il gruppo Amazon Auto EC2 Scaling a un Network Load Balancer. Per ulteriori informazioni, consulta [Collegare un sistema di bilanciamento del carico al gruppo Auto Scaling nella Amazon Auto Scaling](#) User EC2 Guide.

Quindi, puoi aprire la EC2 console Amazon 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 servizi di altro tipo VPC richiedono la seguente configurazione:

- Per Scegli un tipo di destinazione, scegli gli indirizzi IP.
- Per VPC, scegli quello 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 indirizzo, scegli l'indirizzo IP VPC-2.

Per Porte, scegli le tue porte.

- Seleziona Includi come in sospenso di seguito. Quando hai finito di specificare gli indirizzi, scegli Registra obiettivi in sospenso.

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

IPv4indirizzo

172.16.100.60

Porte

80

Fase 4: Creare 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-4i23iuufkdiuflsloi
```

```
us-west-2b - subnet-7x989pkjj78nmn23j
```

```
us-west-2c - subnet-0231ndmas12bnnsds
```

Protocollo/porta

```
TCP : 80
```

Gruppo target

```
my-target-group
```

Fase 5: Creare un servizio VPC endpoint

È possibile utilizzare Network Load Balancer per creare un servizio VPC endpoint. Tramite questo servizio VPC endpoint, Device Farm può connettersi al servizio in VPC-2 senza alcuna infrastruttura aggiuntiva, come un gateway Internet, un'istanza NAT o una VPN connessione.

A tale scopo, consulta [Creazione di un servizio VPC endpoint Amazon](#).

Fase 6: Creare una configurazione VPC degli endpoint in Device Farm

Ora puoi stabilire una connessione privata tra te VPC e Device Farm. È possibile utilizzare Device Farm per testare servizi privati senza esporli attraverso la rete Internet pubblica. A tale scopo, vedere [Creazione di una configurazione di VPC endpoint in Device Farm](#).

Utilizzando lo scenario interregionale di questo argomento, i seguenti valori vengono utilizzati in un esempio di configurazione dell'VPCendpoint:

Nome

```
My VPCE Configuration
```

VPCEnome del servizio

```
com.amazonaws.vpce.us-west-2.vpce-svc-1234567890abcdefg
```

DNSnome del servizio

```
devicefarm.com
```

Fase 7: Creare un test

È possibile creare esecuzioni di test che utilizzano la configurazione dell'VPCendpoint descritta nel [passaggio 6](#). Per ulteriori informazioni, consulta [Crea un'esecuzione di test in Device Farm](#) o [Crea una sessione](#).

Crea una rete scalabile con Transit Gateway

Per creare una rete scalabile utilizzandone più di dueVPCs, puoi utilizzare Transit Gateway per fungere da hub di transito di rete per interconnettere le tue reti VPCs e quelle locali. Per configurare un dispositivo 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 relativi indirizzi IP privati.

Per ulteriori informazioni su Transit Gateway, vedi [Cos'è un gateway di transito?](#) nella Amazon VPC Transit Gateways Guide.

Disattivazione dei dispositivi privati

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.

<Per disattivare un dispositivo privato dopo il periodo iniziale concordato, è n

VPC- ENI in AWS Device Farm

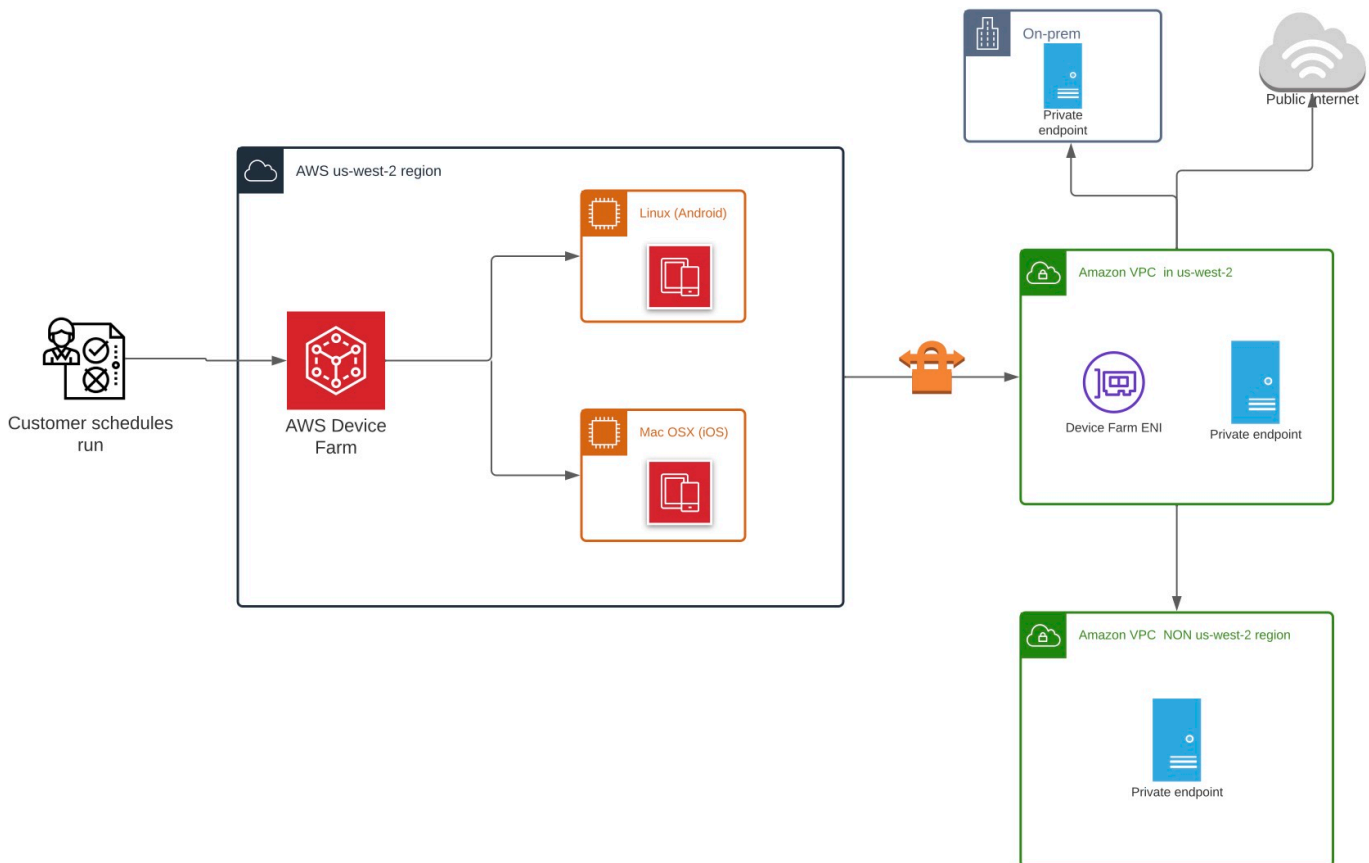
Warning

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 VPC connettività.

AWS La funzionalità di ENI connettività di VPC Device Farm aiuta i clienti a connettersi in modo sicuro ai propri endpoint privati ospitati su AWS software on-premise o su un altro provider di servizi cloud.

Puoi connettere sia i dispositivi mobili Device Farm che le relative macchine host a un ambiente Amazon Virtual Private Cloud (AmazonVPC) 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 privato o non VPC si trova nella us-west-2 regione, puoi collegarlo VPC a un altro della us-west-2 regione utilizzando soluzioni come [Transit Gateway](#) o [VPC Peering](#). In tali situazioni, Device Farm creerà una rete ENI in una sottorete fornita dall'utente per la propria us-west-2 regione VPC e l'utente sarà responsabile di garantire che sia possibile stabilire una connessione tra la us-west-2 regione VPC e l'VPC altra regione.



Per informazioni sull'utilizzo della AWS CloudFormation funzione di creazione automatica e peerVPCs, consulta i [VPC Peering modelli](#) nell'archivio dei AWS CloudFormation modelli su GitHub

Note

Device Farm non addebita alcun costo per la creazione ENIs VPC nell'area del cliente us-west-2. Il costo per l'VPC interconnettività interregionale o esterna non è incluso in questa funzionalità.

Una volta configurato l'VPC accesso, i dispositivi e le macchine host utilizzati per i test non saranno in grado di connettersi a risorse esterne VPC (ad esempio pubbliche CDNs) a meno che non

sia presente un NAT gateway specificato all'VPC interno di. Per ulteriori informazioni, consulta i [NATgateway](#) nella Amazon VPC User Guide.

Argomenti

- [AWS controllo degli accessi e IAM](#)
- [Ruoli collegati ai servizi](#)
- [Prerequisiti](#)
- [Connessione ad Amazon VPC](#)
- [Limiti](#)
- [Utilizzo dei servizi VPC endpoint Amazon con Device Farm - Legacy \(non consigliato\)](#)

AWS controllo degli accessi e IAM

AWS Device Farm consente di utilizzare [AWS Identity and Access Management](#) (IAM) per creare politiche che garantiscono o limitano l'accesso alle funzionalità di Device Farm. Per utilizzare la funzionalità di VPC connettività con AWS Device Farm, è richiesta la seguente IAM politica per l'account utente o il ruolo che si utilizza per accedere a AWS Device Farm:

```
{
  "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 VPC configurazione, la IAM policy deve consentire di richiamare le seguenti azioni sulle risorse elencate nella VPC configurazione:

```
"ec2:DescribeVpcs"
"ec2:DescribeSubnets"
"ec2:DescribeSecurityGroups"
"ec2:CreateNetworkInterface"
```

Inoltre, la IAM policy 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 VPC configurazioni nei propri progetti.

Ruoli collegati ai servizi

AWSDevice Farm utilizza AWS Identity and Access Management (IAM) ruoli [collegati ai servizi](#). Un ruolo collegato al servizio è un tipo di IAM ruolo unico 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 politica di fiducia e la politica di autorizzazione e tale politica di autorizzazione non può essere associata a nessun'altra entità. IAM

È possibile eliminare un ruolo collegato ai servizi solo dopo aver 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 su altri servizi che supportano i ruoli collegati ai servizi, vedi [AWSServizi compatibili con IAM e cerca i servizi con Sì](#) nella colonna Ruolo collegato ai servizi. Scegli Sì in corrispondenza di 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 AWS risorse per conto dell'utente.

Il ruolo `AWSServiceRoleForDeviceFarm` 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
 - Modifica le interfacce di rete
- Per interfacce di rete
 - Crea tag
- Per interfacce EC2 di rete gestite da Device Farm
 - Crea autorizzazioni per l'interfaccia di rete

La IAM politica completa recita:

```
{
```

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

È necessario configurare le autorizzazioni per consentire a un'entità IAM (come un utente, un gruppo o un ruolo) di creare, modificare o eliminare un ruolo collegato al servizio. Per ulteriori informazioni, consulta la sezione [Autorizzazioni relative ai ruoli collegati ai servizi](#) nella Guida per l'utente. IAM

Creazione di un ruolo collegato al servizio per Device Farm

Quando fornisci una VPC configurazione 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 AWS Management Console, o nella AWS CLI AWS API, Device Farm crea il ruolo collegato al servizio per te.

Se elimini questo ruolo collegato ai servizi, puoi 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.

È inoltre possibile utilizzare la IAM console per creare un ruolo collegato al servizio con lo use case Device Farm. In AWS CLI o 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, puoi 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 `AWSServiceRoleForDeviceFarm` collegato al servizio. Dopo aver creato un ruolo collegato al servizio, non potrai modificarne il nome perché varie entità potrebbero farvi riferimento. Tuttavia, è possibile modificare la descrizione del ruolo utilizzando IAM. Per ulteriori informazioni, consulta [Modifica di un ruolo collegato ai servizi nella Guida](#) per l'utente.

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 al servizio utilizzando IAM

Usa la IAM console AWS CLI, o il AWS API per eliminare il ruolo collegato al `AWSServiceRoleForDeviceFarm` servizio. Per ulteriori informazioni, vedere [Eliminazione di un ruolo collegato al servizio nella Guida per l'utente](#). IAM

Regioni supportate per i ruoli collegati al servizio 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 di](#).

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 `AWSServiceRoleForDeviceFarm` ruolo nelle seguenti regioni.

Nome Regione	Identità della regione	Support in Device Farm
US East (N. Virginia)	us-east-1	No
Stati Uniti orientali (Ohio)	us-east-2	No
US West (N. California)	us-west-1	No
US West (Oregon)	us-west-2	Sì
Asia Pacific (Mumbai)	ap-south-1	No
Asia Pacifico (Osaka-Locale)	ap-northeast-3	No
Asia Pacifico (Seul)	ap-northeast-2	No
Asia Pacific (Singapore)	ap-southeast-1	No
Asia Pacific (Sydney)	ap-southeast-2	No

Nome Regione	Identità della regione	Support in Device Farm
Asia Pacifico (Tokyo)	ap-northeast-1	No
Canada (Central)	ca-central-1	No
Europe (Frankfurt)	eu-central-1	No
Europa (Irlanda)	eu-west-1	No
Europe (London)	eu-west-2	No
Europe (Paris)	eu-west-3	No
Sud America (São Paulo)	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 ENI configurazioniVPC:

- I dispositivi privati devono essere assegnati al tuo AWS account.
- È necessario disporre di un AWS account, utente o ruolo con autorizzazioni per creare un ruolo collegato al servizio. Quando si utilizzano gli VPC endpoint Amazon con le funzionalità di test mobile di Device Farm, Device Farm crea un ruolo AWS Identity and Access Management (IAM) collegato al servizio.
- Device Farm può connettersi VPCs solo nella us-west-2 regione. Se non ne hai uno VPC nella us-west-2 regione, devi crearne uno. Quindi, per accedere alle risorse VPC in un'altra regione, devi stabilire una connessione peering tra l'VPCus-west-2area e l'VPCaltra regione. Per informazioni sul peeringVPCs, consulta la [Amazon VPC Peering Guide](#).

È necessario verificare di avere accesso ai dati specificati al VPC momento della configurazione della connessione. È necessario configurare determinate autorizzazioni Amazon Elastic Compute Cloud (AmazonEC2) per Device Farm.

- DNSla risoluzione è richiesta nel file VPC che utilizzi.

- Una volta VPC creato il tuo, avrai bisogno delle seguenti informazioni sulla VPC us-west-2 regione:
 - ID VPC
 - Sottorete IDs
 - Gruppo di sicurezza IDs
- È necessario configurare VPC le connessioni Amazon in base al progetto. Al momento, puoi configurare solo una VPC configurazione per progetto. Quando configuri unVPC, 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 VPC connessione configurata.
- Non è possibile utilizzare VPC - ENI configurazioni insieme alla VPCE funzionalità precedente.
- Si consiglia vivamente di non aggiornare un progetto esistente con una ENI configurazione VPC - poiché i progetti esistenti potrebbero avere VPCE impostazioni che persistono a livello di esecuzione. Invece, se utilizzi già le VPCE funzionalità esistenti, usa VPC - ENI per tutti i nuovi progetti.

Connessione ad Amazon VPC

Puoi configurare e aggiornare il tuo progetto per utilizzare gli VPC endpoint Amazon. La ENI configurazione VPC - è configurata in base al progetto. Un progetto può avere un VPC solo ENI endpoint alla volta. Per configurare VPC l'accesso a un progetto, è necessario conoscere i seguenti dettagli:

- L'VPCID è inserito us-west-2 se la tua app è ospitata lì o l'us-west-2VPCID 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. Si consiglia di associare più sottoreti a diverse zone di disponibilità per migliorare il livello di disponibilità della connettività. VPC

Dopo aver creato la VPC ENI configurazione, è possibile aggiornarne i dettagli utilizzando la console o seguendo la procedura riportata di CLI seguito.

Console

1. Accedere alla console Device Farm all'[indirizzo https://console.aws.amazon.com/devicefarm](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 modificare i VPCSubnets, eSecurity Groups.
6. Selezionare Salva.

CLI

Usa il seguente AWS CLI comando per aggiornare AmazonVPC:

```
$ 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
```

Limiti

Le seguenti limitazioni sono applicabili alla ENI funzionalità VPC -:

- È possibile fornire fino a cinque gruppi di sicurezza nella VPC configurazione di un progetto Device Farm.

- È possibile fornire fino a otto sottoreti nella VPC configurazione di un progetto Device Farm.
- Quando si configura un progetto Device Farm in modo che funzioni con il proprio VPC, la sottorete più piccola che è possibile 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 avete bisogno di un accesso pubblico a Internet durante i test, utilizzate un gateway di [traduzione degli indirizzi di rete \(NAT\)](#). La configurazione di un progetto Device Farm con una sottorete pubblica non fornisce ai test l'accesso a Internet o un indirizzo IP pubblico.
- È supportato solo il traffico in uscita proveniente dal servizio ENI gestito. Ciò significa che ENI non possono ricevere richieste in entrata non richieste da VPC

Utilizzo dei servizi VPC endpoint Amazon con Device Farm - Legacy (non consigliato)

Warning

Consigliamo vivamente di utilizzare la VPC - ENI connectivity descritta in [questa](#) pagina per la connettività privata degli endpoint, poiché ora VPCE è 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 AWS account di utilizzare questa funzionalità con dispositivi privati, [contattaci](#).

Amazon Virtual Private Cloud (AmazonVPC) è un AWS servizio che puoi utilizzare per avviare AWS risorse in una rete virtuale definita dall'utente. Con aVPC, 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 te 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 del tuo dispositivo VPC a Device Farm, puoi utilizzare la VPC console Amazon per creare un servizio VPC endpoint. Questo servizio endpoint consente di fornire la risorsa presente nella Device Farm tramite un VPC endpoint Device Farm. VPC Il servizio endpoint fornisce una connettività affidabile e scalabile a Device Farm senza richiedere un gateway Internet, un'istanza di traduzione degli indirizzi di rete (NAT) o VPN una connessione. Per ulteriori informazioni, consulta [VPC endpoint services \(AWS PrivateLink\) nella Guida](#).AWS PrivateLink

Important

La funzionalità VPC endpoint di Device Farm consente di connettere in modo sicuro i servizi interni privati dell'utente VPC al pubblico di Device Farm VPC utilizzando AWS PrivateLink le connessioni. 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 VPC endpoint in AmazonVPC, puoi utilizzare la console Device Farm per creare una configurazione degli VPC endpoint in Device Farm. Questo argomento mostra come creare la VPC connessione Amazon e la configurazione degli VPC endpoint in Device Farm.

Prima di iniziare

Le seguenti informazioni sono per VPC gli utenti Amazon 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 VPC endpoint con cui è possibile utilizzarlo. Quando crei e configuri un servizio VPC endpoint per l'utilizzo 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 VPC endpoint determina le zone di disponibilità per quel VPC servizio endpoint. Se il servizio VPC endpoint non mostra tutte e tre queste zone di disponibilità, devi ricreare il Network Load Balancer per abilitare queste tre zone e quindi associare nuovamente il Network Load Balancer al servizio endpoint.

- I principali consentiti per il servizio endpoint devono includere l'Amazon Resource Name (ARN) dell'VPC endpoint (servizio) Device Farm. ARN Dopo aver creato il servizio endpoint, aggiungi il servizio VPC endpoint Device Farm ARN all'elenco degli utenti consentiti per concedere a Device Farm l'autorizzazione ad accedere al servizio VPC endpoint. Per ottenere il servizio VPC endpoint Device Farm ARN, [contattateci](#).

Inoltre, se si mantiene attiva l'impostazione Accettazione richiesta quando si crea il servizio VPC endpoint, è necessario 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 VPC console Amazon, 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 di VPC endpoint Amazon che soddisfi questi requisiti.

Fase 1: Creazione di un Network Load Balancer


Il primo passaggio per stabilire una connessione privata tra l'utente 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 (AmazonEC2) 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, scegli Interno.
6. In Network mappings (Mappature di rete), esegui le operazioni seguenti:
 - a. Scegli quello 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 (AmazonEC2) 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, scegli Interno.
6. In Listener, seleziona il protocollo e la porta utilizzati dal gruppo target.
7. In Zone di disponibilità, procedi come segue:

- a. Scegli quella 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 di VPC endpoint Amazon

Dopo aver creato il Network Load Balancer, utilizza la VPC console Amazon per creare un servizio endpoint nel tuo VPC

1. Apri la VPC console Amazon 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 di queste 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 EC2 console Amazon. Segui i passaggi descritti nella [sezione Creazione di un Network Load Balancer](#) a partire dal passaggio 3, quindi continua con questi passaggi nella console AmazonVPC.
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 Crea.
8. Nel nuovo servizio endpoint, scegli Allow principal.
9. [Contattateci](#) per ottenere l'ARNVPCendpoint (servizioARN) Device Farm da aggiungere all'elenco delle autorizzazioni per il servizio endpoint, quindi aggiungete tale servizio ARN all'elenco delle autorizzazioni per il servizio.
10. Nella scheda Details (Dettagli) per il servizio di endpoint, annotare il nome del servizio (service name (nome servizio)). È necessario questo nome quando si crea la configurazione dell'VPCendpoint nel passaggio successivo.

Il tuo servizio VPC endpoint è ora pronto per l'uso con Device Farm.

Fase 3: Creazione di una configurazione VPC degli endpoint in Device Farm

Dopo aver creato un servizio endpoint in AmazonVPC, puoi creare una configurazione di VPC endpoint Amazon in Device Farm.

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

2. Nel riquadro di navigazione, scegli Test per dispositivi mobili, quindi Dispositivi privati.
3. Scegli le VPCEconfigurazioni.
4. Scegli Crea VPCE configurazione.
5. In Crea una nuova VPCE configurazione, inserisci un nome per la configurazione dell'VPCendpoint.
6. Per il nome del VPCE servizio, inserisci il nome del servizio VPC endpoint Amazon (nome del servizio) che hai annotato nella VPC console Amazon. Il nome è simile a `com.amazonaws.vpce.us-west-2.vpce-svc-id`.
7. Per DNSNome del servizio, inserisci il DNS nome del servizio dell'app che desideri testare (ad esempio, `devicefarm.com`). Non specificare `http` o `https` prima del DNS nome del servizio.

Il nome di dominio non è accessibile tramite l'Internet pubblico. Inoltre, questo nuovo nome di dominio, che corrisponde al tuo servizio VPC endpoint, è generato da Amazon Route 53 ed è disponibile esclusivamente per te nella tua sessione Device Farm.

8. Selezionare Salva.

Create a new VPCE configuration ✕

Name
Name of the VPCE configuration.

VPCE service name
Name of the VPCE that will interact with Device Farm VPCE.

Service DNS name
DNS name of your service endpoint. Note: DNS name should not have prefix 'http://' or 'https://'
Example: devicefarm.com

Description - *optional*
Description for the VPCE configuration.

Cancel Save VPCE configuration

Fase 4: Creazione di un test

Dopo aver salvato la configurazione dell'VPCE endpoint, puoi utilizzare la configurazione per creare esecuzioni di test o sessioni di accesso remoto. Per ulteriori informazioni, consulta [Crea un'esecuzione di test in Device Farm](#) o [Crea una sessione](#).

Registrazione delle chiamate API di 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 su un bucket Amazon S3, inclusi eventi per AWS Device Farm. Se non configuri un trail, è comunque possibile visualizzare gli eventi più recenti nella console di CloudTrail in Event history (Cronologia eventi). Utilizzo delle informazioni raccolte da CloudTrail, puoi determinare la richiesta che è stata 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 su CloudTrail, consulta la [Guida per l'utente di AWS CloudTrail](#).

Informazioni su AWS Device Farm in CloudTrail

CloudTrail è abilitato sull'account AWS al momento della sua creazione. Quando si verifica un'attività in AWS Device Farm, tale attività viene registrata in un CloudTrail evento insieme ad altri AWS eventi di servizio in cronologia degli eventi. È possibile visualizzare, cercare e scaricare gli eventi recenti nell'account AWS. Per ulteriori informazioni, consulta [Visualizzazione di eventi mediante la cronologia eventi di CloudTrail](#).

Per una registrazione continua degli eventi nel tuo AWS crea un percorso, inclusi gli eventi per AWS Device Farm. Un percorso abilita CloudTrail per distribuire 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 nella partizione AWS e distribuisce i file di registro nel bucket Amazon S3 specificato. Inoltre, puoi configurare altri servizi AWS per analizzare con maggiore dettaglio e usare i dati raccolti nei log CloudTrail. Per ulteriori informazioni, consulta gli argomenti seguenti:

- [Panoramica della creazione di un percorso](#)
- [Servizi e integrazioni CloudTrail supportati](#)
- [Configurazione delle notifiche Amazon SNS per CloudTrail](#)
- [Ricezione di file di log CloudTrail da più regioni](#) e [Ricezione di file di log CloudTrail da più account](#)

Quando CloudTrail registra la registrazione è abilitata nel tuo 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 AWS record di servizio in un file di registro. CloudTrail determina quando creare e compilare 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 [Riferimento AWS CLI](#) e il [Automazione di Device Farm](#). Ad esempio, le chiamate per creare un nuovo progetto o eseguite in Device Farm generano voci in CloudTrail file di 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 servizio AWS.

Per ulteriori informazioni, consulta [Elemento CloudTrail userIdentity](#).

Comprendere le voci dei file di log di AWS Device Farm

Un percorso è una configurazione che consente la distribuzione di eventi come i file di log in un bucket Amazon S3 specificato. I file di log di CloudTrail contengono una o più voci di log. Un evento rappresenta una singola richiesta da un'fonte e include informazioni sul operazione richiesta, data e ora dell'operazione, parametri richiesti e così via. I file di log di CloudTrail non sono una traccia stack ordinata delle chiamate pubbliche dell'API, quindi non vengono visualizzati in un ordine specifico.

L'esempio seguente mostra un CloudTrail voce di registro che illustra Device Farm ListRuns azione:

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

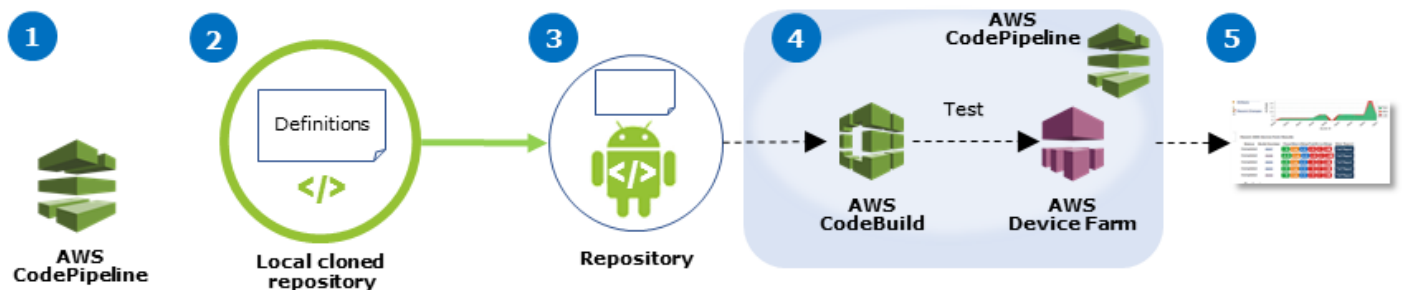
```
}  
]  
}
```

Utilizzo di AWS Device Farm in unCodePipelinefase di test

Puoi usare [AWS CodePipeline](#) per 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 aGitHub](#).

Workflow to Set Up Android Application Test



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

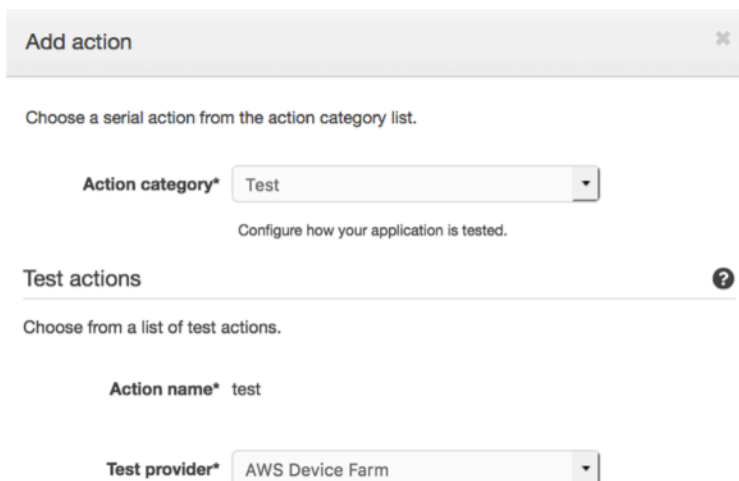
Per imparare a configurare una pipeline che testi in modo continuo un'app compilata (come un file .ipa per iOS o .apk per Android) come sua origine, consulta l'[Esercitazione: testa un'app iOS ogni volta che carichi un file .ipa su un bucket Amazon S3](#).

Configurare CodePipeline per utilizzare i test di Device Farm

In questi passaggi, supponiamo 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 Device Farm, esegui un test come CodePipeline azione di test

1. Accedi a AWS Management Console e apri il CodePipeline console presso <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.



The screenshot shows the 'Add action' dialog in the AWS CodePipeline console. It includes the following fields and options:

- Add action** (dialog title)
- Choose a serial action from the action category list.
- Action category***: Test (dropdown menu)
- Configure how your application is tested.
- Test actions** (section header with a help icon)
- Choose from a list of test actions.
- Action name***: test (text input)
- Test provider***: AWS Device Farm (dropdown menu)

7. Nel Nome del 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.

Device Farm Test

Configure Device Farm test. [Learn more](#)

Project name*	<input type="text" value="DemoProject"/>	<input type="button" value="↻"/>
	↗ Create a new project	
Device pool*	<input type="text" value="Top Devices"/>	<input type="button" value="↻"/>
	↗ Create a new device pool	
App type*	<input type="text" value="iOS"/>	
App file path	<input type="text" value="app-release.apk"/>	
	<small>The location of the application file in your input artifact.</small>	
Test type*	<input type="text" value="Built-in: Fuzz"/>	
Event count	<input type="text" value="6000"/>	
	<small>Specify a number between 1 and 10,000, representing the number of user interface events for the fuzz test to perform.</small>	
Event throttle	<input type="text" value="50"/>	
	<small>Specify a number between 1 and 1,000, representing the number of milliseconds for the fuzz test to wait before performing the next user interface event.</small>	
Randomizer seed	<input type="text"/>	
	<small>Specify a number for the fuzz test to use for randomizing user interface events. Specifying the same number for subsequent fuzz tests ensures identical event sequences.</small>	

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, in Percorso del file di test, immettere il percorso del file di definizione del test. Il percorso è relativo alla cartella principale dell'artefatto di input del test.

The image shows three overlapping screenshots of the AWS Device Farm configuration interface. The top screenshot shows the 'Test type*' dropdown set to 'Calabash' and the 'Test file path' text box containing 'tests.zip'. The middle screenshot shows 'Test type*' set to 'Appium Java TestNG' and 'Appium version' set to '1.7.2'. The bottom screenshot shows 'Test type*' set to 'Built-in: Fuzz', 'Event count' set to '6000', 'Event throttle' set to '50', and a 'Randomizer seed' text box.

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.

▼ Advanced

Device artifacts
Location on the device where custom artifacts will be stored.

Host machine artifacts
Location on the host machine where custom artifacts will be stored.

Add extra data
Location of extra data needed for this test.

Execution timeout
The number of minutes a test run will execute per device before it times out.

Latitude
The latitude of the device expressed in geographic coordinate system degrees.

Longitude
The longitude of the device expressed in geographic coordinate system degrees.

Set Radio Stats

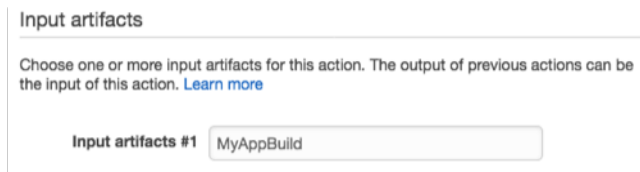
Bluetooth **GPS**

NFC **Wifi**

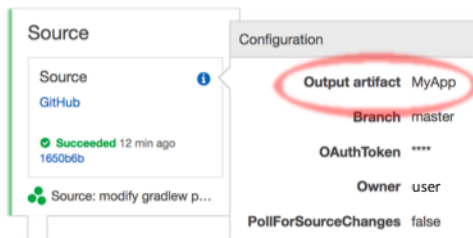
Enable app performance data capture **Enable video recording**

By utilizing on-device testing via Device Farm, you consent to Your Content being transferred to and processed in the United States.

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 console CodePipeline, è possibile individuare il nome dell'artefatto di output per ogni fase passando il puntatore del mouse sull'icona delle informazioni nel diagramma della pipeline. Se la tua pipeline testa la tua app direttamente dalFontefase, scegliMyApp. Se la tua pipeline include unCostruiscipalcoscenico, scegliMyAppBuild.



15. Nella parte inferiore del pannello, scegliere Add Action (Aggiungi operazione).
16. Nel riquadro CodePipeline, scegliere Save pipeline change (Salva modifica alla pipeline) e quindi scegliere Save change (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 ilAWS Command Line Interface(AWS CLI) per eseguire i comandi di Device Farm, vedi[AWS CLI riferimento per AWS Device Farm](#).

Per informazioni generali sulAWS CLI, vedi il[AWS Command Line Interface Guida per l'utente](#)e il[AWS CLI riferimento ai comandi](#).

finestrePowerShellriferimento per AWS Device Farm

Per usare WindowsPowerShellper eseguire i comandi di Device Farm, vedi[Riferimento al cmdlet Device Farm](#)nel[AWS Tools for Windows PowerShellRiferimento al cmdlet](#). Per ulteriori informazioni, vedere[Configurazione degli strumenti AWS per WindowsPowerShell](#)nelAWS Tools for Windows PowerShellGuida per l'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'SDK AWS e l'AWS CLI forniscono i mezzi per farlo.

Il AWS SDK fornisce l'accesso a tutti i servizi AWS, tra cui Device Farm, Amazon S3 e altro ancora. Per ulteriori informazioni, consulta la pagina

- [Strumenti AWS e kit SDK](#)
- [Riferimento all'API AWS Device Farm](#)

Esempio: utilizzo di AWS SDK per avviare una Device Farm, eseguire e raccogliere artefatti

L'esempio seguente fornisce una dimostrazione dall'inizio alla fine di come utilizzare il AWS SDK per funzionare 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.ascii_letters, 4)))

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.

Argomenti

- [Risoluzione dei problemi relativi ai test delle applicazioni Android in AWS Device Farm](#)
- [Risoluzione dei problemi dei JUnit test Java di Appium in Device AWS Farm](#)
- [Risoluzione dei problemi dei test delle applicazioni JUnit web Appium Java in AWS Device Farm](#)
- [Risoluzione dei problemi dei test Appium Java TestNg in Device Farm AWS](#)
- [Risoluzione dei problemi delle applicazioni web Appium Java TestNg in Device Farm AWS](#)
- [Risoluzione dei problemi dei test Appium Python in Device Farm AWS](#)
- [Risoluzione dei problemi dei test delle applicazioni web Appium Python in AWS Device Farm](#)
- [Risoluzione dei problemi relativi ai test di strumentazione in AWS Device Farm](#)
- [Risoluzione dei problemi relativi ai test delle applicazioni iOS in AWS Device Farm](#)
- [Risoluzione dei problemi dei test XCTest in AWS Device Farm](#)
- [Risoluzione dei problemi dei test dell'interfaccia utente XCTest in AWS Device Farm](#)

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 [Utilizzo dei 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-implies-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 [Utilizzo dei 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 [Utilizzo dei 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

Non siamo riusciti a trovare il valore della SDK versione nella tua applicazione. Verifica che l'applicazione sia valida eseguendo il comando `aapt debug badging <path to your test package>` e riprova dopo aver trovato il valore di SDK versione associato alla parola chiave `sdkVersion`.

Durante il processo di convalida del caricamento, AWS Device Farm analizza il valore della SDK versione 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 "sdkVersion"
```

Un pacchetto di un'applicazione Android valido dovrebbe produrre un output come il seguente:

```
sdkVersion:'9'
```

Per ulteriori informazioni, consulta [Utilizzo dei 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'XMLalbero di analisi per un XML file 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 [Utilizzo dei 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 [Utilizzo dei 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 Device AWS 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_PACKAGE_UNZIP_FAILED

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

⚠ Warning

Non siamo riusciti ad aprire il file di test. ZIP 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 [Lavorare con Appium e AWS 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 `dependency-jars` cartella 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 [Lavorare con Appium e AWS Device Farm](#).

**APPIUM_JAVA __ JUNIT _ TEST _ PACKAGE JAR _ MISSING
DEPENDENCY _IN__ DIR**

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

⚠ Warning

Non siamo riusciti a trovare un JAR file nell'albero delle directory dependency-jars. Decomprimi il pacchetto di test, quindi apri la directory dependency-jars, verifica che nella directory sia presente almeno un JAR file 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 .
```

Se il JUnit pacchetto Appium Java è valido, ne troverai almeno uno *jar* file all'interno del *dependency-jars* rubrica:

```
.
|- 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 [Lavorare con Appium e AWS 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, ne troverai almeno uno *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 [Lavorare con Appium e AWS Device Farm](#).

APPIUM_ JAVA _ JUNIT _ TEST __ PACKAGE CLASS _ FILE MISSING TESTS _IN_ _ JAR

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

Warning

Non siamo riusciti a trovare un file di classe all'interno del file dei testJAR. Decomprimi il pacchetto di test e poi decomprimi il JAR file tests, verifica che ci sia almeno un file di classe all'interno del JAR file 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 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 [Lavorare con Appium e AWS 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 JUnit JAR file 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, troverai il file delle JUnit dipendenze simile al file jar *junit-4.10.jar* nel nostro esempio. Il nome deve essere composto dalla parola chiave *junit* e il relativo numero di versione, che in questo esempio è 4.10.

Per ulteriori informazioni, consulta [Lavorare con Appium e AWS 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 suo 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 [Lavorare con Appium e AWS 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 "Appium"](#)

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

Warning

Non siamo riusciti ad aprire il tuo file di testZIP. 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 *dependency-jars* cartella 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 DEPENDENCY _IN_ _ DIR

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

Warning

Non siamo riusciti a trovare un JAR file nell'albero delle directory `dependency-jars`.
Decomprimi il pacchetto di test, quindi apri la directory `dependency-jars`, verifica che nella directory sia presente almeno un JAR file 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 .
```

Se il JUnit pacchetto Appium Java è valido, ne troverai almeno uno *jar* file all'interno del *dependency-jars* rubrica:

```
.
|- 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, ne troverai almeno uno *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 TESTS _IN_ _ JAR

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

Warning

Non siamo riusciti a trovare un file di classe all'interno del file dei testJAR. Decomprimi il pacchetto di test e poi decomprimi il JAR file tests, verifica che ci sia almeno un file di classe all'interno del JAR file 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 almeno un file jar simile *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 JUnit JAR file 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, troverai il file delle JUnit dipendenze simile al file jar *junit-4.10.jar* nel nostro esempio. Il nome deve essere composto dalla parola chiave *junit* e il 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 suo 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 [Lavorare con Appium e AWS Device Farm](#).

Risoluzione dei problemi dei test Appium Java TestNg in Device Farm AWS

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

Non siamo riusciti ad aprire il tuo file di testZIP. 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 [Lavorare con Appium e AWS 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 *dependency-jars* cartella all'interno della cartella 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 [Lavorare con Appium e AWS Device Farm](#).

APPIUM_JAVA __ TESTNG _ TEST _ PACKAGE JAR _ MISSING DEPENDENCY _IN__ DIR

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

Warning

Non siamo riusciti a trovare un JAR file nell'albero delle directory `dependency-jars`. Decomprimi il pacchetto di test, quindi apri la directory `dependency-jars`, verifica che nella directory sia presente almeno un JAR file 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 .
```

Se il JUnit pacchetto Appium Java è valido, ne troverai almeno uno *jar* file all'interno del *dependency-jars* directory.

```
.
|- 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 [Lavorare con Appium e AWS 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, ne troverai almeno uno *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 [Lavorare con Appium e AWS Device Farm](#).

APPIUM_JAVA_TESTNG_TEST__PACKAGE CLASS_FILE MISSING TESTS_IN__JAR

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

Warning

Non siamo riusciti a trovare un file di classe all'interno del file dei testJAR. Decomprimi il pacchetto di test e poi decomprimi il JAR file tests, verifica che ci sia almeno un file di classe all'interno del JAR file 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 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 [Lavorare con Appium e AWS Device Farm](#).

Risoluzione dei problemi delle applicazioni web Appium Java TestNg in Device Farm AWS

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

Non siamo riusciti ad aprire il tuo file di test. ZIP 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 [Lavorare con Appium e AWS Device Farm](#).

APPIUM_WEB_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 *dependency-jars* cartella 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 [Lavorare con Appium e AWS Device Farm](#).

APPIUM_ WEB __ JAVA _ TESTNG _ TEST _ PACKAGE _ JAR MISSING DEPENDENCY _IN__ DIR

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

Warning

Non siamo riusciti a trovare un JAR file nell'albero delle directory dependency-jars. Decomprimi il pacchetto di test, quindi apri la directory dependency-jars, verifica che nella directory sia presente almeno un JAR file 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 .
```

Se il JUnit pacchetto Appium Java è valido, ne troverai almeno uno *jar* file all'interno del *dependency-jars* rubrica.

```
.
|- 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 [Lavorare con Appium e AWS 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, ne troverai almeno uno *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 [Lavorare con Appium e AWS Device Farm](#).

APPIUM_ WEB _ JAVA _ TESTNG _ TEST __ PACKAGE CLASS _ FILE MISSING TESTS _IN_ _ JAR

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

Warning

Non siamo riusciti a trovare un file di classe all'interno del file dei testJAR. Decomprimi il pacchetto di test e poi decomprimi il JAR file tests, verifica che ci sia almeno un file di classe all'interno del JAR file 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 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 [Lavorare con Appium e AWS Device Farm](#).

Risoluzione dei problemi dei test Appium Python in Device Farm AWS

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

Non siamo riusciti ad aprire il tuo file di test Appium. ZIP 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 [Lavorare con Appium e AWS 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, ne troverai almeno uno *.whl* file dipendente come i file evidenziati all'interno di *wheelhouse* directory.

```
.
|-- 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 [Lavorare con Appium e AWS 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, ne troverai almeno uno `.whl` file dipendente come i file evidenziati all'interno di `wheelhouse` directory. Il nome del file può essere diverso, ma dovrebbe terminare con `-any.whl` oppure `-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 [Lavorare con Appium e AWS 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 *tests* cartella 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 [Lavorare con Appium e AWS 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 *tests* cartella 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 [Lavorare con Appium e AWS 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 *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 [Lavorare con Appium e AWS 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 [Lavorare con Appium e AWS 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 [Lavorare con Appium e AWS 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/zhenal/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 [Lavorare con Appium e AWS 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. Controllate la lunghezza del *requirements.txt* file e il numero di *.whl* file dipendenti nella directory Wheelhouse:

```
$ cat requirements.txt | egrep "." | wc -l
    12
$ ls wheelhouse/ | egrep ".+\.whl" | wc -l
    11
```

Se il numero di *.whl* i file dipendenti è inferiore al numero di righe non vuote presenti nel *requirements.txt* file, quindi è necessario assicurarsi di quanto segue:

- C'è un *.whl* file dipendente corrispondente a ciascuna riga del *requirements.txt* file.
- Non ci sono altre righe nel *requirements.txt* file che contiene informazioni diverse dai nomi dei pacchetti di dipendenza.
- Nessun nome di dipendenza viene duplicato su più righe nel *requirements.txt* file in modo tale che due righe del file possano corrispondere a una *.whl* file dipendente.

AWSDevice Farm non supporta le linee in *requirements.txt* file che non corrispondono direttamente ai pacchetti di dipendenza, come le righe che specificano le opzioni globali per il `pip install` comando. Vedi [Formato del file dei requisiti](#) per un elenco di opzioni globali.

Per ulteriori informazioni, consulta [Lavorare con Appium e AWS 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 [Lavorare con Appium e AWS 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, è presente almeno un file correlato *.whl*, come ad esempio 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 [Lavorare con Appium e AWS 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, è presente almeno un file correlato `.whl`, come ad esempio i file evidenziati all'interno della directory `wheelhouse`. Il nome del file potrebbe essere diverso, ma deve 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 [Lavorare con Appium e AWS 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, la directory `tests` si trova 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 [Lavorare con Appium e AWS 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, la directory *tests* si trova all'interno della directory di lavoro. Il nome del file potrebbe essere diverso, ma deve 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 [Lavorare con Appium e AWS Device Farm](#).

APPIUM_WEB_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, il file *requirements.txt* si trova 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 [Lavorare con Appium e AWS 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 .
```

Il file *requirements.txt* dovrebbe trovarsi 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 [Lavorare con Appium e AWS Device Farm](#).

APPIUM_WEB_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS_FAILED

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 [Lavorare con Appium e AWS 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/zhenal/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 [Lavorare con Appium e AWS 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.

INSTRUMENTATION_TEST_PACKAGE_UNZIP_FAILED

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

Warning

Impossibile aprire il file APK 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 `app-debug-androidTest-unaligned.apk`.

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 [Utilizzo della 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.

Warning

Impossibile estrarre informazioni sul pacchetto di test. Verificare che il pacchetto di test sia valido eseguendo il comando "aapt debug badging <percorso del pacchetto di test>" e riprovare quando il comando non stampa più errori.

Durante il processo di convalida del caricamento, Device Farm analizza le informazioni dall'output del comando `aapt debug badging <path to your package>`.

Controllare che questo comando possa essere eseguito correttamente sul pacchetto di test Instrumentation

Nell'esempio seguente, il nome del pacchetto è `app-debug-androidTest-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-implies-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 [Utilizzo della 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.

Warning

Non siamo riusciti a trovare il valore dello instrumentation runner nelAndroidManifest.xml. Verifica che il pacchetto di test sia valido eseguendo il comando «aapt dump xmltree <path to your test package>AndroidManifest.xml», e riprova dopo aver trovato il valore dello instrumentation runner dietro la parola chiave «instrumentation».

Durante il processo di convalida del caricamento, Device Farm analizza il valore dello 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 `app-debug-androidTest-unaligned.apk`.

- 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 [Utilizzo della 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.

Warning

Non siamo riusciti a trovare quello valido `AndroidManifest.xml` nel tuo pacchetto di test. Verifica che il pacchetto di test sia valido eseguendo il comando «`aapt dump xmltree`» `<path to your test package>AndroidManifest.xml`», e riprova dopo che il comando non stampa alcun errore.

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.

Nell'esempio seguente, il nome del pacchetto è `app-debug-androidTest-unaligned.apk`.

- 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 [Utilizzo della strumentazione per Android e AWS Device Farm](#).

INSTRUMENTATION_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

⚠ Warning

Impossibile trovare il nome del pacchetto nel pacchetto di test. Verificare che il pacchetto di test sia valido eseguendo il comando "aapt debug badging <percorso per il pacchetto di test>" e riprovare dopo aver trovato un valore del nome del pacchetto con parola chiave "package: name".

Durante il processo di convalida del caricamento, Device Farm analizza il valore del nome del pacchetto dall'output del comando seguente: `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-androidTest-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 [Utilizzo della 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.

ℹ 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 `AWSDeviceFarmiOSReferenceApp.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 [Utilizzo dei 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 `AWSDeviceFarmiOSReferenceApp.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, la directory *Payload* si trova all'interno della directory di lavoro.

```
.
|-- Payload (directory)
    |-- AWSDeviceFarmiOSReferenceApp.app (directory)
        |-- Info.plist
        |-- (any other files)
```

Per ulteriori informazioni, consulta [Utilizzo dei 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 `AWSDeviceFarmiOSReferenceApp.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 un `.app` directory come `AWSDeviceFarmiOSReferenceApp.app` nel nostro esempio all'interno del *Carico utile* elenco.

```
.
|-- Payload (directory)
    |-- AWSDeviceFarmiOSReferenceApp.app (directory)
        |-- Info.plist
        |-- (any other files)
```

Per ulteriori informazioni, consulta [Utilizzo dei 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 `AWSDeviceFarmiOSReferenceApp.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 *Info.plist* file all'interno di *.app* directory come *AWSDeviceFarmiOSReferenceApp.app* nel nostro esempio.

```
.
|-- Payload (directory)
    |-- AWSDeviceFarmiOSReferenceApp.app (directory)
        |-- Info.plist
        |-- (any other files)
```

Per ulteriori informazioni, consulta [Utilizzo dei 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 e poi apri il file Info.plist all'interno della directory.app, verifica che la chiave «UI»RequiredDeviceCapabilities"è specificato, quindi riprova.

Nell'esempio seguente, il nome del pacchetto è *AWSDeviceFarmiOSReferenceApp.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 .
```

Dovresti trovare il *Info.plist* file all'interno di un *.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 [Utilizzo dei 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 poi apri il file Info.plist all'interno della directory.app, verifica che la chiave «CF»BundleSupportedPlatforms"è specificato, quindi riprova.

Nell'esempio seguente, il nome del pacchetto èAWSDeviceFarmiOSReferenceApp.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 .
```

Dovresti trovare il *Info.plist* file all'interno di un *.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 [Utilizzo dei 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 contiene la parola chiave «simulatore», quindi riprova.

Nell'esempio seguente, il nome del pacchetto è `AWSDeviceFarmiOSReferenceApp.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 .
```

Dovresti trovare il *Info.plist* file all'interno di un *.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 [Utilizzo dei 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 poi apri il file Info.plist all'interno della directory.app, verifica che la chiave «UI»DeviceFamily"è specificato, quindi riprova.

Nell'esempio seguente, il nome del pacchetto èAWSDeviceFarmiOSReferenceApp.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 .
```

Dovresti trovare il *Info.plist* file all'interno di un *.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 [Utilizzo dei 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 poi apri il file Info.plist nella directory.app, verifica che la chiave «CFBundleIdentifier» è specificato, quindi riprova.

Nell'esempio seguente, il nome del pacchetto è `AWSDeviceFarmiOSReferenceApp.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 .
```

Dovresti trovare il `Info.plist` file all'interno di un `.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 [Utilizzo dei 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 poi apri il file Info.plist all'interno della directory.app, verifica che la chiave «CF»BundleExecutable"è specificato, quindi riprova.

Nell'esempio seguente, il nome del pacchetto èAWSDeviceFarmiOSReferenceApp.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 .
```

Dovresti trovare il *Info.plist* file all'interno di un *.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 [Utilizzo dei test iOS in AWS Device Farm](#).

Risoluzione dei problemi dei test XCTest in AWS Device Farm

Il seguente argomento elenca i messaggi di errore che si verificano durante il caricamento di test XCTest 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 `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 .
```

Un pacchetto XCTest valido dovrebbe produrre un output come il seguente:

```
.
|-- swiftExampleTests.xctest (directory)
    |-- Info.plist
    |-- (any other files)
```

Per ulteriori informazioni, consulta [Lavorare con XCTest per iOS e AWS Device Farm](#).

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 pacchetto XCTest è valido, troverai una directory con un nome simile *swiftExampleTests.xctest* all'interno della cartella di lavoro. Il nome deve terminare con *.xctest*.

```
.
|-- swiftExampleTests.xctest (directory)
    |-- Info.plist
    `-- (any other files)
```

Per ulteriori informazioni, consulta [Lavorare con XCTest per iOS e AWS Device Farm](#).

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 pacchetto XCTest è valido, il file *Info.plist* si trova all'interno della directory *.xctest*. Nel nostro esempio seguente, la directory viene chiamata *swiftExampleTests.xctest*.

```
.
|-- swiftExampleTests.xctest (directory)
    |-- Info.plist
    |-- (any other files)
```

Per ulteriori informazioni, consulta [Lavorare con XCTest per iOS e AWS Device Farm](#).

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 e poi apri il file Info.plist, verifica che la chiave «CFBundleIdentifier» è specificato, quindi riprova.

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 .
```

Dovresti trovare il `Info.plist` file all'interno di un `.xctest` directory come `swiftExampleTests.xctest` nel nostro esempio:

```
.
|-- 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 di un'applicazione XCtest valido dovrebbe produrre un output come il seguente:

```
com.amazon.kanapka.swiftExampleTests
```

Per ulteriori informazioni, consulta [Lavorare con XCTest per iOS e AWS Device Farm](#).

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 e poi apri il file Info.plist, verifica che la chiave «CFBundleExecutable» è specificato, quindi riprova.

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 .
```

Dovresti trovare il *Info.plist* file all'interno di un *.xctest* directory come *swiftExampleTests.xctest* nel nostro esempio:

```
.
|-- 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 di un'applicazione XCTest valido dovrebbe produrre un output come il seguente:

```
swiftExampleTests
```

Per ulteriori informazioni, consulta [Lavorare con XCTest per iOS e AWS Device Farm](#).

Risoluzione dei problemi dei test dell'interfaccia utente XCTest in AWS Device Farm

Il seguente argomento elenca i messaggi di errore che si verificano durante il caricamento di test XCTest UI 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.

Warning

Impossibile aprire il file IPA del test. 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 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 [XCTestInterfaccia utente](#).

XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

⚠ Warning

Impossibile trovare la directory Payload nel pacchetto di test. Decomprimere il pacchetto di test, verificare che la directory Payload sia nel pacchetto e riprovare.

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, la directory *Payload* si trova 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 [XCTestInterfaccia utente](#).

XCTEST_UI_TEST_PACKAGE_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 il pacchetto di test, quindi aprire la directory `Payload`, verificare che la directory `.app` si trovi all'interno della directory e riprovare.

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, una directory `.app` come `swift-sampleUITests-Runner.app` del nostro esempio si trova all'interno della directory `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 [XCTestInterfaccia utente](#).

XCTEST_UI_TEST_PACKAGE_PLUGINS_DIR_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

⚠ Warning

Impossibile trovare la directory `Plugins` all'interno della directory `.app`. Decomprimere il pacchetto di test, quindi aprire la directory `.app`, verificare che la directory `Plugins` si trovi all'interno della directory e riprovare.

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, la directory *Plugins* si trova all'interno della directory *.app*. Nell'esempio indicato, la directory si chiama *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 [XCTestInterfaccia utente](#).

XCTEST_UI_TEST_PACKAGE_XCTEST_DIR_MISSING_IN_PLUGINS_DIR

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

⚠ Warning

Impossibile trovare la directory `.xctest` all'interno della directory `plugins`. Decomprimere il pacchetto di test, quindi aprire la directory `plugins`, verificare che la directory `.xctest` si trovi all'interno della directory e riprovare.

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, la directory `.xctest` si trova all'interno della directory `Plugins`. Nell'esempio indicato, la directory si chiama `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 [XCTestInterfaccia utente](#).

XCTEST_UI_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 `.app`. Decomprimere il pacchetto di test, quindi aprire la directory `.app`, verificare che il file `Info.plist` si trovi nella directory e riprovare.

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, il file *Info.plist* si trova all'interno della directory *.app*. Nell'esempio seguente, la directory si chiama *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 [XCTestInterfaccia utente](#).

XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING_IN_XCTEST_DIR

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.

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, il file *Info.plist* si trova all'interno della directory *.xctest*. Nell'esempio seguente, la directory si chiama *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 [XCTestInterfaccia utente](#).

XCTEST_UI_TEST_PACKAGE_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 il pacchetto di test e poi apri il file Info.plist all'interno della directory.app, verifica che la chiave «UI»RequiredDeviceCapabilities"è specificato, quindi riprova.

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 .
```

Il file *Info.plist* dovrebbe trovarsi all'interno di una directory *.app*, come *swift-sampleUITests-Runner.app* del 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 come il seguente:

```
['armv7']
```

Per ulteriori informazioni, consulta [XCTestInterfaccia utente](#).

XCTEST_UI_TEST_PACKAGE_PLATFORM_VALUE_MISSING

In presenza del seguente messaggio, attenersi alla procedura indicata per risolvere il problema.

Warning

Impossibile trovare il valore della piattaforma in Info.plist. Decomprimi il pacchetto di test e poi apri il file Info.plist nella directory.app, verifica che la chiave «CF»BundleSupportedPlatforms"è specificato, quindi riprova.

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 .
```

Il file *Info.plist* dovrebbe trovarsi all'interno di una directory *.app*, come *swift-sampleUITests-Runner.app* del 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 come il seguente:

```
['iPhoneOS']
```

Per ulteriori informazioni, consulta [XCTestInterfaccia utente](#).

XCTEST_UI_TEST_PACKAGE_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 il pacchetto di test e poi apri il file Info.plist nella directory.app, verifica che il valore della chiave «CF»BundleSupportedPlatforms"non contiene la parola chiave «simulatore», quindi riprova.

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 .
```

Il file *Info.plist* dovrebbe trovarsi all'interno di una directory *.app*, come *swift-sampleUITests-Runner.app* del 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)
        |-- (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 come il seguente:

```
['iPhoneOS']
```

Se il pacchetto XCTest UI è valido, il valore non deve contenere la parola chiave: `simulator`.

Per ulteriori informazioni, consulta [XCTestInterfaccia utente](#).

XCTEST_UI_TEST_PACKAGE_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 in Info.plist. Decomprimi il pacchetto di test e poi apri il file Info.plist all'interno della directory.app, verifica che la chiave «UI»DeviceFamily"è specificato, quindi riprova.

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 .
```

Il file *Info.plist* dovrebbe trovarsi all'interno di una directory *.app*, come *swift-sampleUITests-Runner.app* del 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 come il seguente:

```
[1, 2]
```

Per ulteriori informazioni, consulta [XCTestInterfaccia utente](#).

XCTEST_UI_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 e poi apri il file Info.plist nella directory.app, verifica che la chiave «CF»BundleIdentifier"è specificato, quindi riprova.

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 .
```

Il file *Info.plist* dovrebbe trovarsi all'interno di una directory *.app*, come *swift-sampleUITests-Runner.app* del 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)

```

- 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
```

- 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 come il seguente:

```
com.apple.test.swift-sampleUITests-Runner
```

Per ulteriori informazioni, consulta [XCTestInterfaccia utente](#).

XCTEST_UI_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 e poi apri il file Info.plist nella directory.app, verifica che la chiave «CF»BundleExecutable"è specificato, quindi riprova.

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 .
```

Il file *Info.plist* dovrebbe trovarsi all'interno di una directory *.app*, come *swift-sampleUITests-Runner.app* del 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 come il seguente:

```
XCTRunner
```

Per ulteriori informazioni, consulta [XCTestInterfaccia utente](#).

XCTEST_UI_TEST_PACKAGE_TEST_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 all'interno della directory .xctest. Decomprimi il pacchetto di test e poi apri il file Info.plist all'interno della directory.xctest, verifica che la chiave «CFBundleIdentifier» è specificato, e riprova.

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 .
```

Il file *Info.plist* dovrebbe trovarsi all'interno di una directory *.app*, come *swift-sampleUITests-Runner.app* del 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 come il seguente:

```
com.amazon.swift-sampleUITests
```

Per ulteriori informazioni, consulta [XCTestInterfaccia utente](#).

XCTEST_UI_TEST_PACKAGE_TEST_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 all'interno della directory .xctest. Decomprimi il pacchetto di test e poi apri il file Info.plist all'interno della directory.xctest, verifica che la chiave «CFBundleExecutable» è specificato, e riprova.

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 .
```

Il file *Info.plist* dovrebbe trovarsi all'interno di una directory *.app*, come *swift-sampleUITests-Runner.app* del 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 come il seguente:

```
swift-sampleUITests
```

Per ulteriori informazioni, consulta [XCTestInterfaccia utente](#).

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 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 maggiori informazioni sui programmi di conformità applicabili AWS Device Farm, consulta la sezione [AWS Servizi rientranti nell'ambito del programma di conformità AWS](#) .
- **Sicurezza nel cloud:** la tua responsabilità è determinata dal servizio AWS che utilizzi. Sei anche responsabile di altri fattori, tra cui la riservatezza dei dati, i requisiti della tua 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. Scopri anche come utilizzare altri AWS servizi che ti aiutano a monitorare e proteggere le tue risorse Device Farm.

Argomenti

- [Gestione delle identità e degli accessi in AWS Device Farm](#)
- [Convalida della conformità per AWS Device Farm](#)
- [Protezione dei dati in AWS Device Farm](#)
- [Resilienza in AWS Device Farm](#)
- [Sicurezza dell'infrastruttura in AWS Device Farm](#)
- [Analisi e gestione delle vulnerabilità di configurazione in Device Farm](#)
- [Risposta agli incidenti in Device Farm](#)
- [Registrazione e monitoraggio in Device Farm](#)
- [Best practice di sicurezza per Device Farm](#)

Gestione delle identità e degli accessi in AWS Device Farm

Destinatari

Il modo in cui usi AWS Identity and Access Management (IAM) varia a seconda del lavoro svolto in Device Farm.

Utente del servizio: se si utilizza il servizio Device Farm per svolgere il proprio lavoro, l'amministratore fornisce le credenziali e le autorizzazioni necessarie. Man mano che utilizzi più funzionalità di Device Farm per svolgere il tuo lavoro, potresti aver bisogno di autorizzazioni aggiuntive. La comprensione della gestione dell'accesso ti consente di richiedere le autorizzazioni corrette all'amministratore. Se non riesci ad accedere a una funzionalità in Device Farm, consulta [Risoluzione dei problemi relativi all'identità e all'accesso a AWS Device Farm](#).

Amministratore del servizio: se sei responsabile delle risorse di Device Farm presso la tua azienda, probabilmente hai pieno accesso a Device Farm. È tuo compito determinare a quali funzionalità e risorse di Device Farm gli utenti del servizio devono accedere. È quindi necessario inviare richieste all'IAM amministratore per modificare le autorizzazioni degli utenti del servizio. Consulta le informazioni contenute in questa pagina per comprendere i concetti di base di IAM. Per ulteriori informazioni su come la tua azienda può utilizzare IAM Device Farm, consulta [Come funziona AWS Device Farm con IAM](#).

IAM amministratore: se sei un IAM amministratore, potresti voler saperne di più su come scrivere politiche per gestire l'accesso a Device Farm. Per visualizzare esempi di policy basate sull'identità di Device Farm utilizzabili in IAM, vedere [AWS Esempi di policy basate sull'identità di Device Farm](#)

Autenticazione con identità

L'autenticazione è il modo in cui accedi AWS utilizzando le tue credenziali di identità. È necessario autenticarsi (accedere a AWS) come Utente root dell'account AWS, come IAM utente o assumendo un ruolo. IAM

È possibile accedere AWS come identità federata utilizzando le credenziali fornite tramite una fonte di identità. AWS IAM Identity Center Gli utenti (IAM Identity Center), l'autenticazione Single Sign-On della tua azienda e le tue credenziali di Google o Facebook sono esempi di identità federate. Quando accedi come identità federata, l'amministratore aveva precedentemente configurato la federazione delle identità utilizzando i ruoli. IAM Quando si accede AWS utilizzando la federazione, si assume indirettamente un ruolo.

A seconda del tipo di utente, puoi accedere al AWS Management Console o al portale di AWS accesso. Per ulteriori informazioni sull'accesso a AWS, vedi [Come accedere al tuo Account AWS nella Guida per l'Accedi ad AWS utente](#).

Se accedi a AWS livello di codice, AWS fornisce un kit di sviluppo software (SDK) e un'interfaccia a riga di comando () per firmare crittograficamente le tue richieste utilizzando le tue credenziali. CLI Se non utilizzi AWS strumenti, devi firmare tu stesso le richieste. Per ulteriori informazioni sull'utilizzo del metodo consigliato per firmare autonomamente le richieste, consulta [Firmare AWS API le richieste nella Guida per l'IAMutente](#).

A prescindere dal metodo di autenticazione utilizzato, potrebbe essere necessario specificare ulteriori informazioni sulla sicurezza. Ad esempio, ti AWS consiglia di utilizzare l'autenticazione a più fattori (MFA) per aumentare la sicurezza del tuo account. Per ulteriori informazioni, consulta [Autenticazione a più fattori nella Guida per l'AWS IAM Identity Center utente](#) e [Utilizzo dell'autenticazione a più fattori \(MFA\) AWS nella Guida per l'IAMutente](#).

Account AWS utente root

Quando si crea un account Account AWS, si inizia con un'identità di accesso che ha accesso completo a tutte AWS servizi le risorse dell'account. Questa identità è denominata utente Account AWS root ed è accessibile effettuando l'accesso con l'indirizzo e-mail e la password utilizzati per creare l'account. Si consiglia vivamente di non utilizzare l'utente root per le attività quotidiane. Conserva le credenziali dell'utente root e utilizzale per eseguire le operazioni che solo l'utente root può eseguire. Per l'elenco completo delle attività che richiedono l'accesso come utente root, consulta [Attività che richiedono le credenziali dell'utente root](#) nella Guida per l'IAMutente.

IAMUtenti e gruppi

Un [IAMutente](#) è un'identità interna all'utente Account AWS che dispone di autorizzazioni specifiche per una singola persona o applicazione. Laddove possibile, consigliamo di fare affidamento su credenziali temporanee anziché creare IAM utenti con credenziali a lungo termine come password e chiavi di accesso. Tuttavia, se hai casi d'uso specifici che richiedono credenziali a lungo termine con IAM gli utenti, ti consigliamo di ruotare le chiavi di accesso. Per ulteriori informazioni, consulta [Ruotare regolarmente le chiavi di accesso per i casi d'uso che richiedono credenziali a lungo termine](#) nella Guida per l'utente. IAM

Un [IAMgruppo](#) è un'identità che specifica un insieme di utenti. IAM Non è possibile eseguire l'accesso come gruppo. È possibile utilizzare gruppi per specificare le autorizzazioni per più utenti alla volta. I gruppi semplificano la gestione delle autorizzazioni per set di utenti di grandi dimensioni. Ad esempio,

È possibile assegnare un nome a un gruppo IAMAdminse concedere a tale gruppo le autorizzazioni per IAM amministrare le risorse.

Gli utenti sono diversi dai ruoli. Un utente è associato in modo univoco a una persona o un'applicazione, mentre un ruolo è destinato a essere assunto da chiunque ne abbia bisogno. Gli utenti dispongono di credenziali a lungo termine permanenti, mentre i ruoli forniscono credenziali temporanee. Per ulteriori informazioni, consulta [Quando creare un IAM utente \(anziché un ruolo\)](#) nella Guida per l'IAMutente.

Ruoli IAM

Un [IAMruolo](#) è un'identità interna all'utente Account AWS che dispone di autorizzazioni specifiche. È simile a un IAM utente, ma non è associato a una persona specifica. È possibile assumere temporaneamente un IAM ruolo in AWS Management Console [cambiando ruolo](#). È possibile assumere un ruolo chiamando un' AWS APIoperazione AWS CLI or o utilizzando un'operazione personalizzataURL. Per ulteriori informazioni sui metodi di utilizzo dei ruoli, vedere [Utilizzo IAM dei ruoli](#) nella Guida per l'IAMutente.

IAMI ruoli con credenziali temporanee sono utili nelle seguenti situazioni:

- **Accesso utente federato:** per assegnare le autorizzazioni a una identità federata, è possibile creare un ruolo e definire le autorizzazioni per il ruolo. Quando un'identità federata viene autenticata, l'identità viene associata al ruolo e ottiene le autorizzazioni da esso definite. Per informazioni sui ruoli per la federazione, vedere [Creazione di un ruolo per un provider di identità di terze parti](#) nella Guida per l'IAMutente. Se utilizzi IAM Identity Center, configuri un set di autorizzazioni. Per controllare a cosa possono accedere le identità dopo l'autenticazione, IAM Identity Center correla il set di autorizzazioni a un ruolo in. IAM Per informazioni sui set di autorizzazioni, consulta [Set di autorizzazioni](#) nella Guida per l'utente di AWS IAM Identity Center .
- **Autorizzazioni IAM utente temporanee:** un IAM utente o un ruolo può assumere il IAM ruolo di assumere temporaneamente autorizzazioni diverse per un'attività specifica.
- **Accesso su più account:** puoi utilizzare un IAM ruolo per consentire a qualcuno (un responsabile fidato) di un altro account di accedere alle risorse del tuo account. I ruoli sono lo strumento principale per concedere l'accesso multi-account. Tuttavia, con alcuni AWS servizi, è possibile allegare una policy direttamente a una risorsa (anziché utilizzare un ruolo come proxy). Per conoscere la differenza tra ruoli e politiche basate sulle risorse per l'accesso tra account diversi, consulta la [sezione Accesso alle risorse su più account IAM nella Guida per l'utente. IAM](#)
- **Accesso tra servizi:** alcuni AWS servizi utilizzano funzionalità in altri. AWS servizi Ad esempio, quando effettui una chiamata in un servizio, è normale che quel servizio esegua applicazioni

in Amazon EC2 o archivi oggetti in Amazon S3. Un servizio può eseguire questa operazione utilizzando le autorizzazioni dell'entità chiamante, utilizzando un ruolo di servizio o utilizzando un ruolo collegato al servizio.

- **Sessioni di accesso diretto (FAS):** quando utilizzi un IAM utente o un ruolo per eseguire azioni AWS, sei considerato un principale. Quando si utilizzano alcuni servizi, è possibile eseguire un'operazione che attiva un'altra operazione in un servizio diverso. FAS utilizza le autorizzazioni del principale che chiama un AWS servizio, in combinazione con la richiesta di effettuare richieste AWS servizio ai servizi downstream. FAS le richieste vengono effettuate solo quando un servizio riceve una richiesta che richiede interazioni con altri AWS servizi o risorse per essere completata. In questo caso è necessario disporre delle autorizzazioni per eseguire entrambe le azioni. Per i dettagli FAS delle politiche relative alle richieste, consulta [Forward access sessions](#).
- **Ruolo di servizio:** un ruolo di servizio è un [IAM ruolo](#) che un servizio assume per eseguire azioni per conto dell'utente. Un IAM amministratore può creare, modificare ed eliminare un ruolo di servizio dall'interno IAM. Per ulteriori informazioni, vedere [Creazione di un ruolo per delegare le autorizzazioni a un utente AWS servizio nella Guida per l'IAM utente](#).
- **Ruolo collegato al servizio:** un ruolo collegato al servizio è un tipo di ruolo di servizio collegato a un AWS servizio. Il servizio può assumere il ruolo per eseguire un'azione per tuo conto. I ruoli collegati al servizio vengono visualizzati nel tuo account Account AWS e sono di proprietà del servizio. Un IAM amministratore può visualizzare, ma non modificare le autorizzazioni per i ruoli collegati al servizio.
- **Applicazioni in esecuzione su Amazon EC2:** puoi utilizzare un IAM ruolo per gestire le credenziali temporanee per le applicazioni in esecuzione su un'EC2 istanza e che effettuano AWS CLI o richiedono AWS API. È preferibile archiviare le chiavi di accesso all'interno dell'EC2 istanza. Per assegnare un AWS ruolo a un'EC2 istanza e renderlo disponibile per tutte le sue applicazioni, create un profilo di istanza collegato all'istanza. Un profilo di istanza contiene il ruolo e consente ai programmi in esecuzione sull'EC2 istanza di ottenere credenziali temporanee. Per ulteriori informazioni, consulta [Usare un IAM ruolo per concedere le autorizzazioni alle applicazioni in esecuzione su EC2 istanze Amazon nella Guida per l'IAM utente](#).

Per sapere se utilizzare IAM ruoli o IAM utenti, consulta [Quando creare un IAM ruolo \(anziché un utente\)](#) nella Guida per l'IAM utente.

Come funziona AWS Device Farm con IAM

Prima di utilizzare IAM per gestire l'accesso a Device Farm, è necessario comprendere quali IAM funzionalità sono disponibili per l'uso con Device Farm. Per una panoramica generale del

funzionamento di Device Farm e di altri AWS servizi IAM, consulta [AWS Services That Work with IAM](#) nella Guida per l'IAM utente.

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](#)
- [IAM Ruoli di Device Farm](#)

Politiche basate sull'identità di Device Farm

Con le politiche IAM basate sull'identità, è possibile specificare azioni e risorse consentite o negate e le condizioni in base alle quali le azioni sono consentite o negate. Device Farm supporta azioni, risorse e chiavi di condizione specifiche. Per maggiori informazioni su tutti gli elementi utilizzati in una JSON policy, consulta [IAM JSON Policy Elements Reference](#) nella Guida per l'IAM utente.

Azioni

Gli amministratori possono utilizzare AWS JSON le policy per specificare chi ha accesso a cosa. Cioè, quale principale può eseguire operazioni su quali risorse, e in quali condizioni.

L'Action elemento di una JSON policy descrive le azioni che è possibile utilizzare per consentire o negare l'accesso a una policy. Le azioni politiche in genere hanno lo stesso nome dell' AWS API operazione associata. Esistono alcune eccezioni, come le azioni basate solo sulle autorizzazioni che non hanno un'operazione corrispondente. API Esistono anche alcune operazioni che richiedono più operazioni in una policy. Queste operazioni aggiuntive sono denominate operazioni dipendenti.

Includi 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'`CreateTestGridUrl` API operazione 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, vedere [Azioni definite da AWS Device Farm](#) nel IAMService Authorization Reference.

Risorse

Gli amministratori possono utilizzare AWS JSON le policy per specificare chi ha accesso a cosa. Cioè, quale principale può eseguire operazioni su quali risorse, e in quali condizioni.

L'elemento Resource JSON policy specifica l'oggetto o gli oggetti a cui si applica l'azione. Le istruzioni devono includere un elemento Resource o un elemento NotResource. Come best practice, specifica una risorsa utilizzando il relativo [Amazon Resource Name \(ARN\)](#). Puoi eseguire questa operazione per azioni che supportano un tipo di risorsa specifico, note come autorizzazioni a livello di risorsa.

Per le azioni che non supportano le autorizzazioni a livello di risorsa, ad esempio le operazioni di elenco, utilizza un carattere jolly (*) per indicare che l'istruzione si applica a tutte le risorse.

```
"Resource": "*"
```

La risorsa dell'EC2istanza Amazon ha quanto segueARN:

```
arn:${Partition}:ec2:${Region}:${Account}:instance/${InstanceId}
```

Per ulteriori informazioni sul formato diARNs, consulta [Amazon Resource Names \(ARNs\) e AWS Service Namespaces](#).

Ad esempio, per specificare l'i-1234567890abcdf0istanza nella tua dichiarazione, usa quanto segue: 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 EC2 API azioni di Amazon coinvolgono più risorse. Ad esempio, AttachVolume collega un EBS volume Amazon a un'istanza, quindi un IAM utente deve disporre delle autorizzazioni per utilizzare il volume e l'istanza. Per specificare più risorse in una singola istruzione, separale ARNs con virgole.

```
"Resource": [  
    "resource1",  
    "resource2"
```

Per visualizzare un elenco dei tipi di risorse di Device Farm e relativi ARNs, vedere [Tipi di risorse definiti da AWS Device Farm](#) nel IAMService Authorization Reference. Per sapere con quali azioni è possibile specificare il tipo ARN di ciascuna risorsa, vedere [Azioni definite da AWS Device Farm](#) nel IAMService Authorization Reference.

Chiavi di condizione

Gli amministratori possono utilizzare AWS JSON le policy per specificare chi ha accesso a cosa. Cioè, quale principale può eseguire azioni su quali risorse, e in quali condizioni.

L'elemento Condition(o blocco Condition) consente di specificare le condizioni in cui un'istruzione è in vigore. L'elemento Conditionè facoltativo. Puoi 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.

Se specifichi più elementi Conditionin un'istruzione o più chiavi in un singolo elemento Condition, questi vengono valutati da AWS utilizzando un'operazione ANDlogica. Se si specificano più valori per una singola chiave di condizione, AWS valuta la condizione utilizzando un'operazione

logica OR. Tutte le condizioni devono essere soddisfatte prima che le autorizzazioni dell'istruzione vengano concesse.

Puoi anche utilizzare variabili segnaposto quando specifichi le condizioni. Ad esempio, è possibile concedere a un IAM utente l'autorizzazione ad accedere a una risorsa solo se è contrassegnata con il suo nome IAM utente. Per ulteriori informazioni, consulta [gli elementi IAM della politica: variabili e tag](#) nella Guida IAM per l'utente.

AWS supporta chiavi di condizione globali e chiavi di condizione specifiche del servizio. Per visualizzare tutte le chiavi di condizione AWS globali, consulta le chiavi di [contesto delle condizioni AWS globali nella Guida](#) per l'IAM utente.

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 Guida IAM per l'utente.

Per visualizzare un elenco delle chiavi di condizione di Device Farm, vedere [Chiavi di condizione AWS Device Farm](#) nel IAMService Authorization Reference. Per sapere con quali azioni e risorse è possibile utilizzare una chiave di condizione, vedere [Azioni definite da AWS Device Farm](#) nel IAMService Authorization Reference.

Esempi

Per visualizzare esempi di policy basate sull'identità di Device Farm, vedere [AWS Esempi di policy basate sull'identità di 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, fornisci 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, vedere [Etichettatura 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](#).

IAM Ruoli di Device Farm

Un [IAM ruolo](#) 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 per assumere un IAM ruolo o un ruolo tra account. È possibile ottenere credenziali di sicurezza temporanee chiamando AWS STS API operazioni 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 vengono visualizzati nell'IAM account e sono di proprietà del servizio. Un IAM amministratore può visualizzare, ma non modificare, le autorizzazioni per i ruoli collegati al servizio.

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 caratteristica 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 di servizio vengono visualizzati nell'IAM account e sono di proprietà dell'account. Ciò significa che un IAM amministratore può modificare le autorizzazioni per questo ruolo. Tuttavia, il farlo potrebbe pregiudicare la funzionalità del servizio.

Gestione dell'accesso con policy

Puoi controllare l'accesso AWS creando policy e associandole a AWS identità o risorse. Una policy è un oggetto AWS che, se associato a un'identità o a una risorsa, ne definisce le autorizzazioni. AWS valuta queste politiche quando un principale (utente, utente root o sessione di ruolo) effettua una richiesta. Le autorizzazioni nelle policy determinano l'approvazione o il rifiuto della richiesta. La maggior parte delle politiche viene archiviata AWS come JSON documenti. Per ulteriori informazioni sulla struttura e il contenuto dei documenti relativi alle JSON politiche, vedere [Panoramica delle JSON politiche](#) nella Guida per l'IAMutente.

Gli amministratori possono utilizzare AWS JSON le politiche per specificare chi ha accesso a cosa. In altre parole, quale principale può eseguire azioni su quali risorse e in quali condizioni.

Per impostazione predefinita, utenti e ruoli non dispongono di autorizzazioni. Per concedere agli utenti l'autorizzazione a eseguire azioni sulle risorse di cui hanno bisogno, un IAM amministratore può creare IAM politiche. L'amministratore può quindi aggiungere le IAM politiche ai ruoli e gli utenti possono assumerli.

IAMle politiche definiscono le autorizzazioni per un'azione indipendentemente dal metodo utilizzato per eseguire l'operazione. Ad esempio, supponiamo di disporre di una policy che consente l'operazione `iam:GetRole`. Un utente con tale criterio può ottenere informazioni sul ruolo da AWS Management Console, da o da. AWS CLI AWS API

Policy basate su identità

I criteri basati sull'identità sono documenti relativi alle politiche di JSON autorizzazione che è possibile allegare a un'identità, ad esempio un IAM utente, un gruppo di utenti o un ruolo. Tali policy definiscono le azioni che utenti e ruoli possono eseguire, su quali risorse e in quali condizioni. [Per informazioni su come creare una politica basata sull'identità, consulta Creazione di politiche nella Guida per l'utente. IAM IAM](#)

Le policy basate su identità possono essere ulteriormente classificate come policy inline o policy gestite. Le policy inline sono integrate direttamente in un singolo utente, gruppo o ruolo. Le politiche gestite sono politiche autonome che puoi allegare a più utenti, gruppi e ruoli all'interno del tuo Account AWS. Le politiche gestite includono politiche AWS gestite e politiche gestite dai clienti. Per informazioni su come scegliere tra una politica gestita o una politica in linea, consulta [Scelta tra politiche gestite e politiche in linea nella Guida](#) per l'IAMutente.

La tabella seguente descrive le politiche AWS gestite da Device Farm.

Modifica	Descrizione	Data
AWSDeviceFarmFullAccess	Fornisce accesso completo a tutte le operazioni di AWS Device Farm.	15 luglio 2015
AWSServiceRoleForDeviceFarmTestGrid	Consente a Device Farm di accedere alle AWS risorse per tuo conto.	20 maggio 2021

Altri tipi di policy

AWS supporta tipi di policy aggiuntivi e meno comuni. Questi tipi di policy possono impostare il numero massimo di autorizzazioni concesse dai tipi di policy più comuni.

- **Limiti delle autorizzazioni:** un limite di autorizzazioni è una funzionalità avanzata in cui si impostano le autorizzazioni massime che una politica basata sull'identità può concedere a un'entità (utente o ruolo). IAM IAM È possibile impostare un limite delle autorizzazioni per un'entità. Le autorizzazioni risultanti sono l'intersezione delle policy basate su identità dell'entità e i relativi limiti delle autorizzazioni. Le policy basate su risorse che specificano l'utente o il ruolo nel campo `Principal` sono condizionate dal limite delle autorizzazioni. Un rifiuto esplicito in una qualsiasi di queste policy sostituisce l'autorizzazione. [Per ulteriori informazioni sui limiti delle autorizzazioni, consulta Limiti delle autorizzazioni per le entità nella Guida per l'utente. IAM IAM](#)
- **Politiche di controllo del servizio (SCPs):** SCPs sono JSON politiche che specificano le autorizzazioni massime per un'organizzazione o un'unità organizzativa (OU) in. AWS Organizations AWS Organizations è un servizio per il raggruppamento e la gestione centralizzata di più Account AWS di proprietà dell'azienda. Se abiliti tutte le funzionalità di un'organizzazione, puoi applicare le politiche di controllo del servizio (SCPs) a uno o tutti i tuoi account. SCP limita le autorizzazioni per le entità negli account dei membri, inclusa ciascuna Utente root dell'account AWS. Per ulteriori informazioni su Organizations and SCPs, consulta [How SCPs work](#) nella AWS Organizations User Guide.
- **Policy di sessione:** le policy di sessione sono policy avanzate che vengono trasmesse come parametro quando si crea in modo programmatico una sessione temporanea per un ruolo o un utente federato. Le autorizzazioni della sessione risultante sono l'intersezione delle policy basate su identità del ruolo o dell'utente e le policy di sessione. Le autorizzazioni possono anche provenire da una policy basata su risorse. Un rifiuto esplicito in una qualsiasi di queste policy sostituisce

l'autorizzazione. Per ulteriori informazioni, consulta [le politiche di sessione](#) nella Guida IAM per l'utente.

Più tipi di policy

Quando più tipi di policy si applicano a una richiesta, le autorizzazioni risultanti sono più complicate da comprendere. Per informazioni su come AWS determinare se consentire una richiesta quando sono coinvolti più tipi di policy, consulta [Logica di valutazione delle politiche](#) nella Guida per l'IAMutente.

AWSEsempi di policy basate sull'identità di Device Farm

Per impostazione predefinita, IAM gli utenti e i ruoli non dispongono dell'autorizzazione per creare o modificare le risorse di Device Farm. Inoltre, non possono eseguire attività utilizzando AWS Management Console AWS CLI, o AWS API. Un IAM amministratore deve creare IAM politiche che concedano a utenti e ruoli l'autorizzazione a eseguire API operazioni specifiche sulle risorse specifiche di cui ha bisogno. L'amministratore deve quindi allegare tali politiche agli IAM utenti o ai gruppi che richiedono tali autorizzazioni.

Per informazioni su come creare una politica IAM basata sull'identità utilizzando questi documenti di esempioJSON, consulta [Creazione di politiche nella JSON scheda nella Guida per l'utente](#). IAM

Argomenti

- [Best practice per le 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 per le 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 crei o modifichi policy basate su identità, segui 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 ulteriori informazioni, consulta [le politiche AWS gestite o le politiche AWS gestite per le funzioni lavorative](#) nella Guida per l'IAMutente.

- Applica le autorizzazioni con privilegi minimi: quando imposti le autorizzazioni con le IAM politiche, concedi solo le autorizzazioni necessarie per eseguire un'attività. Puoi farlo definendo le azioni che possono essere intraprese su risorse specifiche in condizioni specifiche, note anche come autorizzazioni con privilegi minimi. Per ulteriori informazioni sull'utilizzo per applicare le autorizzazioni, consulta [Politiche](#) e autorizzazioni nella Guida IAM per l'utente. IAM IAM
- Utilizza le condizioni nelle IAM politiche per limitare ulteriormente l'accesso: puoi aggiungere una condizione alle tue politiche per limitare l'accesso ad azioni e risorse. Ad esempio, puoi scrivere una condizione di policy per specificare che tutte le richieste devono essere inviate utilizzando SSL. È inoltre possibile utilizzare condizioni per concedere l'accesso alle azioni di servizio se vengono utilizzate tramite uno specifico AWS servizio, ad esempio AWS CloudFormation. Per ulteriori informazioni, consulta [Elementi IAM JSON della politica: Condizione](#) nella Guida IAM per l'utente.
- Usa IAM Access Analyzer per convalidare IAM le tue policy e garantire autorizzazioni sicure e funzionali: IAM Access Analyzer convalida le policy nuove ed esistenti in modo che aderiscano al linguaggio delle IAM policy () e alle best practice. JSON IAM IAMAccess Analyzer fornisce più di 100 controlli delle politiche e consigli pratici per aiutarti a creare policy sicure e funzionali. Per ulteriori informazioni, vedere [Convalida delle policy di IAM Access Analyzer nella Guida per l'utente. IAM](#)
- Richiedi l'autenticazione a più fattori (MFA): se hai uno scenario che richiede l'utilizzo di IAM utenti o di un utente root Account AWS, attiva questa opzione MFA per una maggiore sicurezza. Per richiedere MFA quando vengono richiamate API le operazioni, aggiungi MFA delle condizioni alle tue politiche. Per ulteriori informazioni, consulta [Configurazione dell'APIaccesso MFA protetto nella Guida](#) per l'IAMutente.

Per ulteriori informazioni sulle procedure consigliate in IAM, consulta la sezione [Procedure consigliate in materia di sicurezza IAM nella](#) Guida per l'IAMutente.

Consentire agli utenti di visualizzare le loro autorizzazioni

Questo esempio mostra come è possibile creare una politica che consenta IAM agli utenti di visualizzare le politiche in linea e gestite allegate alla loro identità utente. Questa politica include le autorizzazioni per completare questa azione sulla console o utilizzando o a livello di codice. AWS CLI AWS API

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupForUser",
        "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 IAM utente 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`.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "GetTestGridProject",
      "Effect": "Allow",
      "Action": [
        "devicefarm:GetTestGridProject"
      ],
      "Resource": "arn:aws:devicefarm:us-west-2:111122223333:testgrid-
project:123e4567-e89b-12d3-a456-426655441111"
    },
    {
      "Sid": "ViewProjectInfo",
      "Effect": "Allow",
      "Action": [
        "devicefarm:ListTestGridSessions",
        "devicefarm:ListTestGridSessionActions",
        "devicefarm:ListTestGridSessionArtifacts"
      ],
      "Resource": "arn:aws:devicefarm:us-west-2:111122223333:testgrid-*:123e4567-
e89b-12d3-a456-426655441111/*"
    }
  ]
}
```

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 IAM politica delinea una politica 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:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "$id": "scheduleTestRuns",
      "effect": "Allow",
      "Action": [ "devicefarm:CreateUpload", "devicefarm:ScheduleRun" ],

```

```

    "Resource": [
      "arn:aws:devicefarm:us-west-2:111122223333:project:123e4567-e89b-12d3-
a456-426655440000",
      "arn:aws:devicefarm:us-west-2:111122223333:*:123e4567-e89b-12d3-
a456-426655440000/*",
    ]
  }
]
}

```

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 `Owner` della risorsa richiesta corrisponde al nome utente dell'account richiedente.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ListTestGridProjectSessions",
      "Effect": "Allow",
      "Action": [
        "devicefarm:ListTestGridSession*",
        "devicefarm:GetTestGridSession",
        "devicefarm:ListTestGridProjects"
      ],
      "Resource": [
        "arn:aws:devicefarm:us-west-2:testgrid-project:*/*",
        "arn:aws:devicefarm:us-west-2:testgrid-session:*/*"
      ],
      "Condition": {
        "StringEquals": {"aws:TagKey/Owner": "${aws:username}"}
      }
    }
  ]
}

```

Puoi allegare questa politica agli IAM utenti del tuo account. Se un utente denominato `richard-roe` tenta di visualizzare un progetto o una sessione di Device Farm, il progetto deve essere taggato `Owner=richard-roe` o `owner=richard-roe`. In caso contrario, a questo utente viene negato l'accesso. La chiave di tag di condizione `Owner` corrisponde sia a `Owner` che a `owner` perché i nomi delle chiavi di condizione non distinguono tra maiuscole e minuscole. Per ulteriori informazioni, consulta [Elementi IAM JSON della politica: Condizione](#) nella Guida IAM per l'utente.

Risoluzione dei problemi relativi all'identità e all'accesso a 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 AWS Management Console 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.

L'errore di esempio seguente si verifica quando l'IAM utente tenta di utilizzare la console per visualizzare i dettagli di un'esecuzione, ma non dispone di `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 AWS servizi 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.

L'errore di esempio seguente si verifica quando un IAM utente 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 passare 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 assistenza, contatta AWS l'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 IAM utente, è possibile visualizzare l'ID della 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 perdi la chiave di accesso segreta, devi aggiungere nuove chiavi di accesso all'IAMutente. 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 IAM per l'utente.

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 IAM le politiche ai IAM ruoli associati alla persona o all'applicazione. Per ulteriori informazioni, vedere [Set di autorizzazioni](#) nella Guida per l'AWS IAM Identity Center utente.

Se non si utilizza IAM Identity Center, è necessario creare IAM entità (utenti o ruoli) per le persone o le applicazioni che necessitano di accesso. È quindi necessario allegare una policy 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. AWS Per ulteriori informazioni sulla creazione di IAM utenti, gruppi, politiche e autorizzazioni, consulta [IAM identità](#) e [politiche e autorizzazioni IAM nella](#) Guida per l'utente. IAM

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 ulteriori informazioni, consulta gli argomenti seguenti:

- Per sapere se Device Farm supporta queste funzionalità, vedere [Come funziona AWS Device Farm con IAM](#).
- Per informazioni su Account AWS come fornire l'accesso alle risorse di tua proprietà, consulta [Fornire l'accesso a un IAM utente di un altro Account AWS utente di tua proprietà](#) nella Guida per l'IAMutente.
- Per scoprire come fornire l'accesso alle tue risorse a terze parti Account AWS, consulta [Fornire l'accesso a persone Account AWS di proprietà di terzi](#) nella Guida per l'IAMutente.

- Per informazioni su come fornire l'accesso tramite la federazione delle identità, consulta [Fornire l'accesso agli utenti autenticati esternamente \(federazione delle identità\)](#) nella Guida per l'IAMutente.
- Per conoscere la differenza tra l'utilizzo di ruoli e politiche basate sulle risorse per l'accesso tra account diversi, consulta la sezione Accesso alle [risorse tra account nella Guida per l'utente](#). IAM 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 PCIRAMP, Fed HIPAA e altri. AWS Device Farm non rientra nell'ambito di alcun programma di AWS conformità.

Per un elenco dei AWS servizi che rientrano nell'ambito di specifici programmi di conformità, vedere [AWS Servizi compresi nell'ambito del programma di conformità AWS](#) . Per informazioni generali, vedere Programmi di [AWS conformità Programmi](#) di di .

È possibile scaricare report di audit di terze parti utilizzando AWS Artifact. Per ulteriori informazioni, consulta [Scaricamento dei report in AWS Artifact](#) Scaricamento dei . AWS

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 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](#)— 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 AWS servizi utilizzati. Per ulteriori informazioni sulla privacy dei dati, consulta la sezione [Privacy dei dati FAQ](#). Per informazioni sulla protezione dei dati in Europa, consulta il [Modello di responsabilitàAWS condivisa e GDPR](#) il post sul blog sulla AWS sicurezza.

Ai fini della protezione dei dati, ti consigliamo di proteggere Account AWS le credenziali e di configurare i singoli utenti con AWS IAM Identity Center o AWS Identity and Access Management (IAM). In tal modo, a ogni utente verranno assegnate solo le autorizzazioni necessarie per svolgere i suoi compiti. Ti suggeriamo, inoltre, di proteggere i dati nei seguenti modi:

- Utilizza l'autenticazione a più fattori (MFA) con ogni account.
- Usa SSL/TLS per comunicare con AWS le risorse. Richiediamo TLS 1.2 e consigliamo TLS 1.3.
- Configurazione API e registrazione delle attività degli utenti con AWS CloudTrail.
- Utilizza soluzioni di AWS crittografia, insieme a tutti i controlli di sicurezza predefiniti all'interno AWS servizi.
- 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 FIPS 140-2 moduli crittografici convalidati per accedere AWS tramite un'interfaccia a riga di comando o un'API, usa un endpoint. FIPS Per ulteriori informazioni sugli FIPS endpoint disponibili, vedere [Federal Information Processing Standard \(\) 140-2. FIPS](#)

Ti consigliamo vivamente 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 AWS servizi utilizzando la console API, AWS CLI, o AWS SDKs. I dati inseriti nei tag o nei campi di testo in formato libero utilizzati per i nomi possono essere utilizzati per la fatturazione o i log di diagnostica. Se fornisci un URL a un server esterno, ti consigliamo vivamente di non includere le informazioni sulle credenziali URL per convalidare la tua richiesta a quel server.

Crittografia in transito

Gli endpoint Device Farm supportano solo richieste firmate HTTPS (SSL/TLS), tranne dove diversamente indicato. Tutti i contenuti recuperati o inseriti in Amazon S3 tramite URLs caricamento vengono crittografati SSL utilizzando/. TLS Per ulteriori informazioni su come vengono HTTPS effettuate le richieste AWS, consulta [Firmare AWS API le richieste](#) 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 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 dallo storage 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 al dispositivo privato, il dispositivo viene rimosso dall'uso e distrutto in modo sicuro in conformità alle politiche di AWS distruzione. Per ulteriori informazioni, consulta [Utilizzo di 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 VPC endpoint Amazon per connettersi alle tue risorse in 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 AmazonVPC. Indipendentemente dalla configurazione degli VPC endpoint, Device Farm isola il traffico dagli altri utenti della 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 globale di AWS è basata su Regioni e zone di disponibilità AWS. AWS Le Regioni forniscono più zone di disponibilità fisicamente separate e isolate che sono connesse tramite reti altamente ridondanti, a bassa latenza e velocità effettiva elevata. Con le Zone di disponibilità, è possibile progettare e gestire applicazioni e database che eseguono il failover automatico 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.

Per ulteriori informazioni sulle Regioni AWS e sulle zone di disponibilità, consulta [Infrastruttura globale di AWS](#).

Perché Device Farm è disponibile in us-west-2 Solo per regione, si consiglia vivamente di implementare i 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.

API Le chiamate AWS pubblicate vengono utilizzate per accedere a Device Farm attraverso la rete. I client devono supportare quanto segue:

- Transport Layer Security (TLS). Richiediamo TLS 1.2 e consigliamo TLS 1.3.
- Suite di cifratura con Perfect Forward Secrecy (PFS) come (Ephemeral Diffie-Hellman) o DHE (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 O puoi 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 VPC endpoint Amazon per proteggere le connessioni in entrata e in uscita dal tuo AWS account.

Sicurezza dell'infrastruttura per i 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 attraverso l'endpoint generato con `HTTPS createTestGridUrl`.

Al momento, la funzionalità di test del browser desktop non supporta la configurazione VPC degli endpoint di Amazon. L'utente ha la responsabilità di assicurarsi che ogni istanza di test di Device Farm abbia accesso sicuro alle risorse da essa testate.

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 scansione il grafico delle dipendenze alla ricerca di vulnerabilità note, vedere Rilevamento delle vulnerabilità di sicurezza su GitHub](#)
- Per informazioni sul correttore di dipendenza di Open Web Application Security Project (OWASP) Maven, consultate il sito Web. [OWASP DependencyCheck](#)OWASP

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 AWS servizi, 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, vedere [Registrazione delle chiamate API di 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, gestiscile come considerazioni utili anziché prescrizioni.

- Concedi a qualsiasi sistema di integrazione continua (CI) in IAM cui utilizzi il minimo privilegio possibile. 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 Guida per l'[IAMutente](#).
- 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 [Lavorare con ambienti di test personalizzati](#)

Limiti in AWS Device Farm

L'elenco seguente descrive gli attuali limiti di AWS Device Farm:

- Le dimensioni massime di un file di un'app che è possibile caricare è di 4 GB.
- 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 incrementato su richiesta.
- Non vi è alcun limite al numero di esecuzioni che puoi pianificare.
- La durata massima di una sessione di accesso remoto è di 150 minuti.
- La durata massima di un'esecuzione di test automatizzata è di 150 minuti.
- Il numero massimo di lavori in corso, compresi i lavori in coda in sospeso sull'account, è 250. Si tratta di un limite flessibile.
- Non vi è alcun limite al numero di dispositivi che puoi includere in un'esecuzione di test. Il numero di dispositivi o processi su cui è possibile eseguire 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 su AWS Device Farm è 5. Puoi richiedere un aumento di questo numero 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.

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 di 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 AWS Device Farm con il plug-in Jenkins CI](#)
- [Plugin AWS Device Farm Gradle](#)

Integrazione di AWS Device Farm con il plug-in Jenkins CI

Questo plugin fornisce la funzionalità di 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 [GitHub](#) e segui le istruzioni contenute in [Fase 1: Installazione del plugin](#).

Questa sezione contiene una serie di procedure per configurare e utilizzare il plugin Jenkins CI con AWS Device Farm.

Argomenti

- [Fase 1: Installazione del plugin](#)
- [Fase 2: Creare unAWS Identity and Access Managementutente per il tuo Jenkins CI Plugin](#)
- [Fase 3: Istruzioni per la prima configurazione](#)
- [Fase 4: Usare il plugin in un job Jenkins](#)
- [Dipendenze](#)

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](#)

Project Hello World App

[Workspace](#)

[Recent Changes](#)

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](#)



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](#)


Post-build Actions

Run Tests on AWS Device Farm

refresh

Project 

[Required] Select your AWS Device Farm project.

Device Pool 

[Required] Select your AWS Device Farm device pool.

Application 

[Required] Pattern to find newly built application.

Store test results locally.

Choose test to run

- Built-in Fuzz
- Appium Java JUnit
- Appium Java TestNG
- Calabash

Features 

[Required] Pattern to find features.zip.

Tags 

[Optional] Tags to pass into Calabash.

- Instrumentation
- Android UI Automator

Delete

Add post-build action ▼

Save

Apply

Il plugin può anche aprire tutti gli artefatti del test (log, screenshot, ecc.) in locale:



Jenkins > Hello World App > #19

- Back to Project
- Status
- Changes
- Console Output
- Edit Build Information
- Delete Build
- AWS Device Farm
- Previous Build

Artifacts of Hello World App #19

 [AWS Device Farm Results](#) /

-  [Amazon Kindle Fire HDX 7 \(WiFi\)](#)
-  [Motorola DROID Ultra \(Verizon\)](#)
-  [Samsung Galaxy Note 4 \(AT&T\)](#)
-  [Samsung Galaxy S5 \(AT&T\)](#)
-  [Samsung Galaxy Tab 4 10.1 Nook \(WiFi\)](#)

 [\(all files in zip\)](#)

Fase 1: Installazione del plugin

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. Scarica 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: Creare unAWS Identity and Access Managementutente per il tuo Jenkins CI Plugin

Ti consigliamo di non utilizzare ilAWSaccount root per accedere a Device Farm. Invece, crea un nuovoAWS Identity and Access Managementutente (IAM) (o utilizza un utente IAM esistente) nel tuoAWSaccount, quindi accedi a Device Farm con quell'utente IAM.

Per creare un nuovo utente IAM, vedi[Creazione di un utente IAM \(AWS Management Console\)](#).

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.

Concedi all'utente IAM l'autorizzazione 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

LaAWSaccount 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. Selezionare Policies (Policy).
3. Scegliere Create Policy (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:

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

8. Scegliere Create Policy (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 policy 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: Istruzioni per la prima configurazione

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 a AWS Device Farm installazione.
5. Copia le tue credenziali di sicurezza da [Fase 2: Creare un utente IAM](#) e incolla il tuo ID chiave di accesso e la tua chiave di accesso segreta nelle rispettive caselle.
6. Seleziona 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'installazione Post-build Actions (Operazioni post-compilazione).
5. Fare clic su **Aggiungi un'azione post-costruzione** e seleziona **Esegui test su 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. Seleziona Salva.

Dipendenze

Il plugin Jenkins CI richiede l'SDK AWS Mobile 1.10.5 o versione successiva. Per ulteriori informazioni e per installare l'SDK, consulta [SDK AWS Mobile](#).

Plugin 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

Per scaricare il plugin Gradle, vai a [GitHub](#) segui le istruzioni in [Creazione del plug-in Device Farm Gradle](#).

Il plugin Device Farm Gradle fornisce 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 plugin Device Farm Gradle.

Argomenti

- [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](#)
- [Fase 4: Configurazione dei tipi di test](#)
- [Dipendenze](#)

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 a [GitHub](#) e clona il repository.
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 plug-in 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](#)

Fase 3: Generazione di un utente IAM

AWS Identity and Access Management(IAM) ti aiuta a gestire le autorizzazioni e le politiche con cui lavorareAWSrisorse. Questo argomento illustra come generare un utente IAM con le 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.

Ti consigliamo di non utilizzare ilAWSaccount root per accedere a Device Farm. Invece, crea un nuovo utente IAM (o usa un utente IAM esistente) nelAWSaccount, quindi accedi a Device Farm con quell'utente IAM.

Note

IlAWSaccount 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. Seleziona Create (Crea).
6. Scegliere Download Credentials (Scarica credenziali) e salvarle in una posizione in cui saranno facilmente recuperabili in seguito.
7. Scegli Close (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. ScegliClicca quidove c'è scritto,Non ci sono politiche in linea da mostrare. Per crearne una, fai clic qui.

11. Sulla schermata Set Permissions (Imposta autorizzazioni), selezionare Custom Policy (Personalizza policy).
12. Selezionare Select (Seleziona).
13. Assegnare un nome alla policy, ad esempio **AWSDeviceFarmGradlePolicy**.
14. Incollare la seguente policy in Policy Document (Documento policy).

```
{
  "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, vedere [Creazione di un utente IAM \(AWS Management Console\)](#) o [Configurazione](#).

Fase 4: Configurazione dei tipi di test

Per impostazione predefinita, il plugin AWS Device Farm Gradle esegue il [Utilizzo della 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, vedere [Utilizzo dei tipi di test 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 [Integrato: fuzz \(Android e iOS\)](#).

Instrumentation

Device Farm fornisce supporto per la strumentazione (EspressoJUnit, Robotium o qualsiasi test basato sulla strumentazione) per Android. Per ulteriori informazioni, consulta [Utilizzo della strumentazione per Android e AWS Device Farm](#).

Quando si esegue un test di strumentazione in Gradle, Device Farm utilizza il .apk file generato dalla androidTestdirectory come fonte dei test.

```
instrumentation {  
  
    filter "test filter per developer docs" // optional  
  
}
```

Dipendenze

Runtime

- Il plugin Device Farm Gradle richiede ilAWSMobile SDK 1.10.15 o versione successiva. 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

Cronologia dei documenti

La tabella seguente descrive le modifiche importanti apportate alla documentazione dall'ultima versione della guida.

Modifica	Descrizione	Data della modifica
supporto AL2	Device Farm ora supporta l'ambiente di test AL2 per Android. Scopri di più 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 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 .	15 maggio 2023
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: <ul style="list-style-type: none">• Appium (Python)• Appium (Python)	20 aprile 2020
Nuove informazioni di sicurezza e sul tagging delle risorse AWS.	Per semplificare e ampliare la messa in sicurezza dei servizi AWS, è stata creata una nuova sezione sulla sicurezza. Per ulteriori informazioni, consulta Sicurezza in AWS Device Farm	27 marzo 2020

Modifica	Descrizione	Data della modifica
	È stata aggiunta una nuova sezione sull'etichettatura in Device Farm. Per ulteriori informazioni sul tagging, consulta Etichettatura in Device Farm .	
Rimozione di Direct Device Access.	L'accesso diretto ai dispositivi (debug remoto su dispositivi 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 configurazione Gradle, con parametri opzionali commentati. Ulteriori informazioni su Configurazione del plug-in Device Farm Gradle .	16 agosto 2019
Nuovo requisito per le sessioni di test con XCTest	Per le esecuzioni di test che utilizzano il framework XCTest, 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 Utilizzo dei tipi di test in AWS Device Farm .	10 gennaio 2019

Modifica	Descrizione	Data della modifica
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 Utilizzo dei tipi di test 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 Lavorare con ambienti di test personalizzati .	16 agosto 2018
Supporto 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 Utilizzo di AWS Device Farm in unCodePipelinefase di test .	19 luglio 2018

Modifica	Descrizione	Data della modifica
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 Utilizzo di dispositivi privati in AWS Device Farm .	2 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 Imposta il 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 Simula le connessioni e le condizioni di rete per il funzionamento 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
Sessioni di accesso remoto	È ora possibile accedere da remoto e interagire con un singolo dispositivo nella console. Ulteriori informazioni su Lavorare con l'accesso remoto .	19 aprile 2016
Slot di dispositivi self-service	Ora puoi acquistare slot di dispositivi utilizzando la AWS Management Console, il AWS Command Line Interface oppure l'API. Ulteriori informazioni su come Acquista uno slot per dispositivi in Device Farm .	22 marzo 2016

Modifica	Descrizione	Data della modifica
Come arrestare sessioni di test	È ora possibile arrestare le sessioni di test utilizzando la AWS Management Console, il AWS Command Line Interface oppure l'API. Ulteriori informazioni su come Interrompere un'esecuzione in AWS Device Farm .	22 marzo 2016
Nuovi tipi di test per XCTest UI	È ora possibile eseguire i test personalizzati per XCTest UI su applicazioni iOS. Ulteriori informazioni sul tipo di test XCTestInterfaccia utente .	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 Utilizzo dei tipi di test in AWS Device Farm .	19 gennaio 2016
Tipi di test per applicazioni Web	È ora possibile eseguire test personalizzati per Appium Java JUnit e TestNG su applicazioni Web. Ulteriori informazioni su Utilizzo dei test delle 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 analizzando ogni schermata come se fosse un utente finale e acquisendo screenshot durante l'analisi.	16 settembre 2015
Supporto iOS aggiunto	Ulteriori informazioni sulla verifica dei dispositivi iOS e l'esecuzione di test iOS (tra cui XCTest) in Utilizzo dei tipi di test 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

Glossario per AWS

Per la terminologia AWS più recente, consultare il [glossario AWS](#) nella documentazione di riferimento per 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.