



Guida per gli sviluppatori

AWS Device Farm



Versione API 2015-06-23

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Device Farm: Guida per gli sviluppatori

Copyright © 2025 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 utente IAM nel tuo AWS account	4
Passaggio 3: concedere all'utente IAM l'autorizzazione 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
Passaggio 3: Creare e avviare una corsa	8
Passaggio 4: Visualizza i risultati della corsa	10
Passaggi successivi	11
Acquisto di slot per dispositivi	12
Acquista slot per dispositivi (console)	12
Acquista uno slot per dispositivo (AWS CLI)	14
Acquista uno slot per dispositivo (API)	18
Annullamento dello slot di un dispositivo	18
Annullare lo slot di un dispositivo (console)	18
Annullare lo slot di un dispositivo (AWS CLI)	19
Annullare lo slot di un dispositivo (API)	19
Concetti	20
Dispositivi	20
Dispositivi supportati	21
Pool di dispositivi	21
Dispositivi privati	21
Branding del dispositivo	21
Slot per dispositivi	21
App preinstallate per dispositivi	22
Funzionalità del dispositivo	22

Ambienti di test	22
Ambiente di test standard	23
Ambiente di test personalizzato	23
Esecuzioni	23
Esegui la configurazione	24
Esegui la conservazione dei file	24
Esegui lo stato del dispositivo	24
Esecuzioni parallele	24
Impostazione del timeout di esecuzione	24
Annunci nelle puntate	25
File multimediali in tirature	25
Attività comuni per le esecuzioni	25
App	25
App di strumentazione	25
Riassegnare la firma delle app durante le esecuzioni	25
App offuscate in corso di esecuzione	26
Report	26
Conservazione dei report	26
Componenti dei report	26
Registra i report di accesso	27
Attività comuni per i report	27
Sessioni	27
Dispositivi supportati per l'accesso remoto	27
Conservazione dei file di sessione	27
App di strumentazione	28
Riassegnare la firma delle app nelle sessioni	28
App offuscate nelle sessioni	28
Progetti	29
Creare un progetto	29
Prerequisiti	29
Crea un progetto (console)	29
Crea un progetto (AWS CLI)	30
Crea un progetto (API)	30
Visualizzazione dell'elenco dei progetti	30
Prerequisiti	31
Visualizza l'elenco dei progetti (console)	31

Visualizza l'elenco dei progetti (AWS CLI)	31
Visualizza l'elenco dei progetti (API)	31
Esecuzioni di test	32
Creazione di un'esecuzione di test	32
Prerequisiti	33
Crea un'esecuzione di test (console)	33
Crea un test run (AWS CLI)	36
Creare un'esecuzione di test (API)	46
Passaggi successivi	46
Impostazione del timeout di esecuzione	47
Prerequisiti	47
Imposta il timeout di esecuzione per un progetto	48
Imposta il timeout di esecuzione per un'esecuzione di test	48
Simulazione di connessioni e condizioni di rete	48
Imposta la configurazione della rete quando pianifichi un'esecuzione di test	49
Crea un profilo di rete	49
Modifica le condizioni della rete durante il test	51
Interrompere una corsa	51
Interrompi una corsa (console)	51
Interrompi una corsa (AWS CLI)	53
Interrompere un'esecuzione (API)	55
Visualizzazione di un elenco di esecuzioni	55
Visualizza un elenco di esecuzioni (console)	55
Visualizza un elenco di esecuzioni (AWS CLI)	55
Visualizzare un elenco di esecuzioni (API)	56
Creazione di 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
Visualizzazione dei rapporti sui test	59
Scaricamento di artefatti	69
Inserimento di tag in Device Farm	74
Applicazione di tag alle risorse	74
Ricerca di risorse per tag	75

Rimozione dei tag dalle risorse	76
Framework di test e test integrati	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
Integrazione con i test Appium	79
Test Android	94
Framework di test delle applicazioni Android	94
Tipi di test integrati per Android	94
Instrumentation	94
Test iOS	98
Framework di test delle applicazioni iOS	98
Tipi di test integrati per iOS	98
XCTest	98
XCTest UI	101
Test delle app Web	104
Regole per i dispositivi misurati e non misurati	104
Test integrati	105
Integrato: fuzz (Android e iOS)	105
Ambienti di test personalizzati	107
Sintassi delle specifiche di test	108
Esempio di specifiche di test	110
Ambiente di test Android	116
Software supportato	117
Intervalli IP supportati per l'ambiente di test Amazon Linux 2	118
Utilizzo dello devicefarm-cli strumento	119
Selezione di un host di test Android	120
Esempio di file di specifiche di test	122
Migrazione ad Amazon Linux 2 Test Host	126
Variabili di ambiente	128

Variabili di ambiente comuni	129
Variabili di ambiente Java Appium JUnit	131
Variabili di ambiente Appium Java TestNg	131
XCUITest variabili di ambiente	131
Migrazione dei test	132
Considerazioni sulla migrazione	132
Fasi della migrazione	133
Struttura Appium	134
Strumentazione Android	134
Migrazione dei test iOS XCUITest esistenti	134
Estensione della modalità personalizzata	134
Impostazione del PIN del dispositivo	135
Accelerazione dei test basati su Appium	136
Utilizzo di Webhook e altro APIs	139
Aggiungere file aggiuntivi al pacchetto di test	140
Accesso remoto	143
Creazione di una sessione	143
Prerequisiti	144
Creare una sessione con la console Device Farm	144
Passaggi successivi	144
Utilizzo di una sessione	145
Prerequisiti	145
Utilizzare una sessione nella console Device Farm	145
Passaggi successivi	146
Suggerimenti e trucchi	146
Recupero dei risultati delle sessioni dalle sessioni di accesso remoto	147
Prerequisiti	147
Visualizzazione dei dettagli della sessione	147
Scaricamento del video o dei registri della sessione	147
Dispositivi privati	148
Creazione di un profilo dell'istanza	149
Richiedi dispositivi privati aggiuntivi	151
Creazione di un'esecuzione di test o di una sessione di accesso remoto	153
Selezione di dispositivi privati	154
Regole ARN del dispositivo	155
Regole delle etichette delle istanze del dispositivo	156

Regole ARN dell'istanza	156
Crea un pool di dispositivi privato	157
Creazione di un pool di dispositivi privato con dispositivi privati (AWS CLI)	159
Creazione di un pool di dispositivi privati con dispositivi privati (API)	160
Ignorare la rifirma dell'app	160
Salta la rifirma dell'app sui dispositivi Android	161
Ignora la nuova firma dell'app sui dispositivi iOS	161
Crea una sessione di accesso remoto per rendere affidabile la tua app	162
Amazon VPC in tutte le regioni	164
Panoramica del peering VPC per diverse regioni VPCs	164
Prerequisiti per l'utilizzo di Amazon VPC	165
Stabilire una connessione peering tra due VPCs	166
Aggiornamento delle tabelle di routing per VPC-1 e VPC-2	167
Creazione di gruppi target	167
Creazione di un Network Load Balancer	169
Creazione di un servizio endpoint VPC	170
Creazione di una configurazione di endpoint VPC nell'applicazione	170
Creazione di un'esecuzione di test	171
Creazione di sistemi VPC scalabili	171
Disattivazione di dispositivi privati in Device Farm	171
Connettività VPC	172
AWS controllo degli accessi e IAM	174
Ruoli collegati ai servizi	175
Autorizzazioni di ruolo collegate ai servizi per Device Farm	176
Creazione di un ruolo collegato al servizio per Device Farm	179
Modifica di un ruolo collegato al servizio per Device Farm	179
Eliminazione di un ruolo collegato al servizio per Device Farm	179
Regioni supportate per i ruoli collegati al servizio Device Farm	180
Prerequisiti	181
Connessione ad Amazon VPC	182
Limiti	183
Utilizzo dei servizi endpoint VPC - Legacy	184
Prima di iniziare	185
Fase 1: Creazione di un Network Load Balancer	186
Fase 2: Creare un servizio endpoint VPC	188
Fase 3: Creare una configurazione di endpoint VPC	189

Fase 4: Creare un'esecuzione di test	191
Registrazione delle chiamate API di AWS CloudTrail con	192
Informazioni su AWS Device Farm in CloudTrail	192
Informazioni sulle voci dei file di log di AWS Device Farm	193
Integrazione con AWS Device Farm	196
Configura CodePipeline per utilizzare i test Device Farm	197
AWS CLI riferimento	201
PowerShell Riferimento per Windows	202
Automazione di Device Farm	203
Esempio: utilizzo dell' AWS SDK per avviare un'esecuzione di Device Farm e raccogliere artefatti	203
Risoluzione dei problemi	208
Risoluzione dei problemi relativi alle applicazioni Android	208
ANDROID_APP_UNZIP_FAILED	208
ANDROID_APP_AAPT_DEBUG_BADGING_FAILED	209
ANDROID_APP_PACKAGE_NAME_VALUE_MISSING	210
ANDROID_APP_SDK_VERSION_VALUE_MISSING	211
ANDROID_APP_AAPT_DUMP_XMLTREE_FAILED	212
ANDROID_APP_DEVICE_ADMIN_PERMISSIONS	213
Alcune finestre della mia applicazione Android mostrano una schermata vuota o nera	215
Risoluzione dei problemi di Appium Java JUnit	215
APPIUM_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED	215
APPIUM_JAVA_JUNIT_TEST_PACKAGE_DEPENDENCY_DIR_MISSING	216
APPIUM_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DIR	217
APPIUM_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING	218
APPIUM_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JAR	219
APPIUM_JAVA_JUNIT_TEST_PACKAGE_JUNIT_VERSION_VALUE_UNKNOWN	221
APPIUM_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION	222
Risoluzione dei problemi Web Appium Java JUnit	224
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED	224
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_DEPENDENCY_DIR_MISSING	225
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DIR ..	226
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING	227
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JAR ..	228
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_JUNIT_VERSION_VALUE_UNKNOWN ...	229
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION	230

Risoluzione dei problemi di Appium Java TestNg	231
APPIUM_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED	232
APPIUM_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSING	233
APPIUM_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DIR	234
APPIUM_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING	235
APPIUM_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JAR	236
Risoluzione dei problemi web Appium Java TestNg	237
APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED	237
APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSING	238
APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DIR	239
APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING	240
APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JAR	241
Risoluzione dei problemi di Appium Python	243
APPIUM_PYTHON_TEST_PACKAGE_UNZIP_FAILED	243
APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING	244
APPIUM_PYTHON_TEST_PACKAGE_INVALID_PLATFORM	245
APPIUM_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING	246
APPIUM_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME	247
APPIUM_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISSING	248
APPIUM_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION	249
APPIUM_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS_FAILED	251
APPIUM_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED	252
APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEELS_INSUFFICIENT	253
Risoluzione dei problemi web di Appium Python	254
APPIUM_WEB_PYTHON_TEST_PACKAGE_UNZIP_FAILED	254
APPIUM_WEB_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING	255
APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PLATFORM	256
APPIUM_WEB_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING	258
APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME	259
APPIUM_WEB_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISSING	260
APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION	261
APPIUM_WEB_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS_FAILED	262
APPIUM_WEB_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED	263
Risoluzione dei problemi relativi ai test di strumentazione	265
INSTRUMENTATION_TEST_PACKAGE_UNZIP_FAILED	265
INSTRUMENTATION_TEST_PACKAGE_AAPT_DEBUG_BADGING_FAILED	266

INSTRUMENTATION_TEST_PACKAGE_INSTRUMENTATION_RUNNER_VALUE_MISSING	267
INSTRUMENTATION_TEST_PACKAGE_AAPT_DUMP_XMLTREE_FAILED	268
INSTRUMENTATION_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSING	269
Risoluzione dei problemi relativi alle applicazioni iOS	270
IOS_APP_UNZIP_FAILED	270
IOS_APP_PAYLOAD_DIR_MISSING	271
IOS_APP_APP_DIR_MISSING	272
IOS_APP_PLIST_FILE_MISSING	273
IOS_APP_CPU_ARCHITECTURE_VALUE_MISSING	274
IOS_APP_PLATFORM_VALUE_MISSING	275
IOS_APP_WRONG_PLATFORM_DEVICE_VALUE	276
IOS_APP_FORM_FACTOR_VALUE_MISSING	278
IOS_APP_PACKAGE_NAME_VALUE_MISSING	279
IOS_APP_EXECUTABLE_VALUE_MISSING	280
Risoluzione dei problemi XCTest	282
XCTEST_TEST_PACKAGE_UNZIP_FAILED	282
XCTEST_TEST_PACKAGE_XCTEST_DIR_MISSING	283
XCTEST_TEST_PACKAGE_PLIST_FILE_MISSING	283
XCTEST_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING	284
XCTEST_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING	286
Risoluzione dei problemi XCTest dell'interfaccia	287
XCTEST_UI_TEST_PACKAGE_UNZIP_FAILED	287
XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_MISSING	288
XCTEST_UI_TEST_PACKAGE_APP_DIR_MISSING	289
XCTEST_UI_TEST_PACKAGE_PLUGINS_DIR_MISSING	290
XCTEST_UI_TEST_PACKAGE_XCTEST_DIR_MISSING_IN_PLUGINS_DIR	291
XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING	292
XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING_IN_XCTEST_DIR	292
XCTEST_UI_TEST_PACKAGE_CPU_ARCHITECTURE_VALUE_MISSING	293
XCTEST_UI_TEST_PACKAGE_PLATFORM_VALUE_MISSING	295
XCTEST_UI_TEST_PACKAGE_WRONG_PLATFORM_DEVICE_VALUE	296
XCTEST_UI_TEST_PACKAGE_FORM_FACTOR_VALUE_MISSING	297
XCTEST_UI_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING	299
XCTEST_UI_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING	300
XCTEST_UI_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSING	301
XCTEST_UI_TEST_PACKAGE_TEST_EXECUTABLE_VALUE_MISSING	303

XCTEST_UI_TEST_PACKAGE_MULTIPLE_APP_DIRS	304
XCTEST_UI_TEST_PACKAGE_MULTIPLE_IPA_DIRS	305
XCTEST_UI_TEST_PACKAGE_BOTH_APP_AND_IPA_DIR_PRESENT	306
XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_PRESENT_IN_ZIP	307
Sicurezza	309
Gestione dell'identità e degli accessi	310
Destinatari	310
Autenticazione con identità	310
Come funziona AWS Device Farm con IAM	314
Gestione dell'accesso con policy	319
Esempi di policy basate su identità	321
Risoluzione dei problemi	326
Convalida della conformità	329
Protezione dei dati	330
Crittografia in transito	331
Crittografia a riposo	331
Conservazione dei dati	331
Gestione dei dati	332
Gestione delle chiavi	333
Riservatezza del traffico Internet	333
Resilienza	333
Sicurezza dell'infrastruttura	334
Sicurezza dell'infrastruttura per il test dei dispositivi fisici	334
Sicurezza dell'infrastruttura per il test dei browser desktop	335
Analisi della configurazione e delle vulnerabilità	335
Risposta agli incidenti	336
Registrazione di log e monitoraggio	337
Best practice di sicurezza	337
Limiti	338
Strumenti e plugin	340
Plug-in Jenkins CI	340
Dipendenze	343
Installazione del plugin Jenkins CI	343
Creazione di un utente IAM per il tuo plugin Jenkins CI	344
Configurazione del plugin Jenkins CI per la prima volta	346
Utilizzo del plugin in un job Jenkins	346

Plugin Device Farm Gradle	347
Dipendenze	347
Creazione del plugin Device Farm Gradle	348
Configurazione del plugin Device Farm Gradle	348
Generazione di un utente IAM nel plugin Device Farm Gradle	351
Configurazione dei tipi di test	353
Cronologia dei documenti	355
AWS Glossario	360
.....	ccclxi

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 nella regione us-west-2 (Oregon).

Test automatizzato delle app

Device Farm consente di caricare i propri test o 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, viene aggiornato un rapporto di test che contiene risultati di alto livello, log di basso livello, pixel-to-pixel schermate e dati sulle prestazioni.

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 [Framework di test e test integrati 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 la pagina dei [prezzi di AWS Device Farm](#).

project

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

report

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

run

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

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 utente IAM nel tuo AWS account](#)
- [Passaggio 3: concedere all'utente IAM l'autorizzazione 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.

Parte della procedura di registrazione prevede la ricezione di una telefonata o di un messaggio di testo e l'immissione di un codice di verifica sulla tastiera del telefono.

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

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

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

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

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

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

Note

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

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

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

Per ulteriori informazioni sulla creazione di policy IAM, consulta [Creating IAM Policies](#) nella IAM User Guide.

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

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

Approfondimenti

Ora sei pronto per iniziare a usare Device Farm. Consultare [Guida introduttiva a Device Farm](#).

Guida introduttiva a Device Farm

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

Note

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

Argomenti

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

Prerequisiti

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

- Completa le fasi descritte in [Configurazione](#). Sono necessari un AWS account e un utente AWS Identity and Access Management (IAM) con autorizzazione per accedere a Device Farm.
- Per Android, puoi importare un file.apk (pacchetto app Android) o utilizzare l'applicazione di esempio che forniamo. Per iOS, è necessario disporre di un file .ipa (archivio di app iOS). Caricherai il file su Device Farm più avanti in questa procedura dettagliata.

Note

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

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

Fase 1: Accesso alla console di

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

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

Fase 2: Creare un progetto

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

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

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

Passaggio 3: Creare e avviare una corsa

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

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

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

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

Note

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

Note

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

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

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



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



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

Passaggio 4: Visualizza i risultati della corsa

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

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

- Una sezione per scaricare il risultato dell'analisi.

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

Passaggi successivi

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

Acquisto di uno slot per dispositivi in Device Farm

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

Acquista slot per dispositivi (console)

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

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

Important

Se modifichi il numero di slot per dispositivi ma visualizzi il messaggio «Contattaci» o «Contattaci per l'acquisto», significa che il tuo AWS account non è ancora autorizzato ad acquistare il numero di slot per dispositivi che hai richiesto.

Queste opzioni richiedono di inviare un'e-mail al team di supporto di Device Farm. Nell'e-mail, specifica il numero di ogni tipo di dispositivo che desideri acquistare e per quale ciclo di fatturazione.

Note

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

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

Versione API 2015-06-23 13

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

Acquista uno slot per dispositivo (AWS CLI)

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

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

```
{
  "accountSettings": {
    "maxSlots": {
      "GUID": 1,
      "GUID": 1,
      "GUID": 1,
      "GUID": 1
    },
    "unmeteredRemoteAccessDevices": {
      "ANDROID": 0,
      "IOS": 0
    },
    "maxJobTimeoutMinutes": 150,
    "trialMinutes": {
      "total": 1000.0,
      "remaining": 954.1
    },
    "defaultJobTimeoutMinutes": 150,
    "awsAccountNumber": "AWS-ACCOUNT-NUMBER",
    "unmeteredDevices": {
      "ANDROID": 0,
      "IOS": 0
    }
  }
}
```

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

```
{
```

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

```

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

```

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

Note

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

Verrà visualizzato un output simile al seguente:

```

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

```

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

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

```
}
```

Per questa funzionalità sono disponibili anche i `list-offering-transactions` comandi `renew-offering` and. Per ulteriori informazioni, consulta la [AWS CLI riferimento](#).

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 dell'API Device Farm, vedere [Automazione di Device Farm](#).

Annullamento dello slot di un dispositivo in Device Farm

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

Per ulteriori informazioni sugli slot dei dispositivi, consulta. [Acquisto di uno slot per dispositivi in Device Farm](#)

Annullare lo slot di un dispositivo (console)

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

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

Annullare lo slot di un dispositivo (AWS CLI)

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

Annullare lo slot di un dispositivo (API)

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

Concetti di AWS Device Farm

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

Questa sezione descrive importanti concetti di Device Farm.

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

Per ulteriori informazioni sui tipi di test supportati in Device Farm, vedere [Framework di test e test integrati in AWS Device Farm](#).

Supporto per dispositivi in AWS Device Farm

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

Argomenti

- [Dispositivi supportati](#)
- [Pool di dispositivi](#)
- [Dispositivi privati](#)
- [Branding del dispositivo](#)
- [Slot per dispositivi](#)
- [App preinstallate per dispositivi](#)
- [Funzionalità del dispositivo](#)

Dispositivi supportati

Device Farm fornisce supporto per centinaia di dispositivi Android e iOS unici e popolari e combinazioni di sistemi operativi. L'elenco dei dispositivi disponibili cresce con l'immissione sul mercato di nuovi dispositivi. Per l'elenco completo dei dispositivi, consulta [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 [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. I dispositivi non sono connessi con alcun operatore e non possono effettuare chiamate né inviare SMS.

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

Google Play Services 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 [Ambienti di test personalizzati in AWS Device Farm](#).

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

Per ulteriori informazioni, consulta [Caricamento di una specifica di test personalizzata utilizzando and](#). AWS CLI [Creazione di un'esecuzione di test in Device Farm](#)

Funziona in AWS Device Farm

Le seguenti sezioni contengono informazioni sulle esecuzioni in Device Farm.

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

Argomenti

- [Esegui la configurazione](#)

- [Esegui la conservazione dei file](#)
- [Esegui lo stato del dispositivo](#)
- [Esecuzioni parallele](#)
- [Impostazione del timeout di esecuzione](#)
- [Annunci nelle puntate](#)
- [File multimediali in tirature](#)
- [Attività comuni per le esecuzioni](#)

Esegui la configurazione

Come parte di un'esecuzione, è possibile fornire impostazioni che Device Farm può utilizzare per sovrascrivere le impostazioni correnti del dispositivo. Queste includono le coordinate di latitudine e longitudine, le impostazioni locali, gli stati delle radiofrequenze (ad esempio, Bluetooth, GPS, NFC e Wi-Fi), dati supplementari (contenuti in un file .zip) e app ausiliarie (app che dovrebbero essere installate prima di testare l'app).

Esegui la conservazione dei file

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

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

Esegui lo stato del dispositivo

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

Esecuzioni parallele

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

Impostazione del timeout di esecuzione

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

Per ulteriori informazioni, consulta [Impostazione del timeout di esecuzione per le esecuzioni di test in AWS Device Farm](#).

Annunci nelle puntate

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

File multimediali in tirature

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

Attività comuni per le esecuzioni

Per ulteriori informazioni, consulta [Creazione di un'esecuzione di test in Device Farm](#) e [I test vengono eseguiti in AWS Device Farm](#).

App in AWS Device Farm

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

Argomenti

- [App di strumentazione](#)
- [Riassegnare la firma delle app durante le esecuzioni](#)
- [App offuscate in corso di esecuzione](#)

App di strumentazione

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

Riassegnare la firma delle app durante le esecuzioni

Per le app iOS, non è necessario aggiungere alcun Device Farm UUIDs al profilo di provisioning. Device Farm sostituisce il profilo di provisioning incorporato con un profilo wildcard e quindi firma nuovamente l'app. Se fornisci dati ausiliari, Device Farm li aggiunge al pacchetto dell'app prima

che Device Farm li installi, in modo che l'ausiliario esista nella sandbox dell'app. La nuova firma dell'app rimuove diritti come App Group, Associated Domains, Game Center,, Wireless Accessory Configuration HealthKit, In-App Purchase HomeKit, Inter-App Audio, Apple Pay, Notifiche push e Configurazione e controllo VPN.

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

App offuscate in corso di esecuzione

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

Report in AWS Device Farm

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

Argomenti

- [Conservazione dei report](#)
- [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 [Visualizzazione dei report dei test in Device Farm](#).

Sessioni in AWS Device Farm

È possibile utilizzare Device Farm per eseguire test interattivi delle 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 rifirma le app Android e iOS. Ciò può causare l'interruzione della funzionalità che dipende dalla firma dell'app. Ad esempio, l'API di Google Maps per Android dipende dalla firma della tua app. La nuova firma delle app può inoltre attivare il rilevamento delle norme antipirateria o antimanomissione da parte di prodotti come quelli per dispositivi Android. DexGuard

App offuscate nelle sessioni

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

Progetti in AWS Device Farm

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

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

Argomenti

- [Creazione di un progetto in AWS Device Farm](#)
- [Visualizzazione dell'elenco dei progetti in AWS Device Farm](#)

Creazione di un progetto in AWS Device Farm

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

Prerequisiti

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

Crea un progetto (console)

1. Accedere alla console Device Farm all'indirizzo <https://console.aws.amazon.com/devicefarm>.
2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.
3. Scegliere New project (Nuovo progetto).
4. Inserisci un nome per il 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 [Impostazione del 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 AWS CLI risposta include l'Amazon Resource Name (ARN) del progetto.

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

Per ulteriori informazioni, consulta [create-project](#) e [AWS CLI riferimento](#).

Crea un progetto (API)

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

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

Visualizzazione dell'elenco dei progetti in AWS Device Farm

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

Argomenti

- [Prerequisiti](#)
- [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 [Creazione di un progetto in AWS Device Farm](#), poi torna a questa pagina.

Visualizza l'elenco dei progetti (console)

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

Visualizza l'elenco dei progetti (AWS CLI)

- Per visualizzare l'elenco dei progetti, esegui il comando [list-projects](#).

Per visualizzare informazioni su un singolo progetto, esegui il comando [get-project](#).

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

Visualizza l'elenco dei progetti (API)

- Per visualizzare l'elenco dei progetti, chiama l'API [ListProjects](#).

Per visualizzare informazioni su un singolo progetto, chiama l'API [GetProject](#).

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

I test vengono eseguiti in AWS Device Farm

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

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

Argomenti

- [Creazione di un'esecuzione di test in Device Farm](#)
- [Impostazione del timeout di esecuzione per le esecuzioni di test in AWS Device Farm](#)
- [Simulazione delle connessioni e delle condizioni di rete per le esecuzioni di AWS Device Farm](#)
- [Interruzione di un'esecuzione in AWS Device Farm](#)
- [Visualizzazione di un elenco di esecuzioni in AWS Device Farm](#)
- [Creazione di un pool di dispositivi in AWS Device Farm](#)
- [Analisi dei risultati dei test in AWS Device Farm](#)

Creazione di un'esecuzione di test in Device Farm

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

Argomenti

- [Prerequisiti](#)
- [Crea un'esecuzione di test \(console\)](#)
- [Crea un test run \(AWS CLI\)](#)
- [Creare un'esecuzione di test \(API\)](#)
- [Passaggi successivi](#)

Prerequisiti

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

Crea un'esecuzione di test (console)

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

Note

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

Per informazioni sulle suite di test disponibili, consulta [Framework di test e test integrati in AWS Device Farm](#).

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

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

14. (Facoltativo) Per aggiungere una configurazione aggiuntiva, apri il menu a discesa Configurazione aggiuntiva. In questa sezione, puoi effettuare una delle seguenti operazioni:
 - Per fornire altri dati da utilizzare a Device Farm durante l'esecuzione, accanto a Aggiungi dati aggiuntivi, scegli Scegli file, quindi cerca e scegli il file.zip che contiene i dati.
 - Per installare un'app aggiuntiva da utilizzare in Device Farm durante l'esecuzione, accanto a Installa altre app, scegli Scegli file, quindi cerca e scegli il file.apk o .ipa che contiene l'app. Ripetere questa operazione per altre applicazioni che si desidera installare. Puoi modificare l'ordine di installazione trascinando le app dopo averle caricate.

- Per specificare se una rete Wi-Fi, Bluetooth, GPS o NFC sia abilitata durante la sessione, accanto a Set radio states (Imposta stati radio), selezionare le caselle appropriate.
- Per preconfigurare la latitudine e la longitudine del dispositivo per la sessione, accanto a Device location (Posizione dispositivo), immettere le coordinate.
- Per preimpostare le impostazioni locali del dispositivo per l'esecuzione, in Impostazioni locali del dispositivo, scegli le impostazioni locali.
- Seleziona Abilita la registrazione video per registrare video durante il test.
- Seleziona Abilita l'acquisizione dei dati sulle prestazioni dell'app per acquisire i dati sulle prestazioni dal dispositivo.

Note

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

Note

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

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

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



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



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

Se desideri interrompere l'esecuzione del test, consulta [Interruzione di un'esecuzione in AWS Device Farm](#).

Crea un test run (AWS CLI)

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

Argomenti

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

Fase 1: Scegli un progetto

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

1. Per elencare i tuoi progetti Device Farm, esegui `list-projects`. Se non disponi ancora di un progetto, consulta [Creazione di un progetto in AWS Device Farm](#).

Esempio:

```
aws devicefarm list-projects
```

La risposta include un elenco dei tuoi progetti Device Farm.

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

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

Passaggio 2: Scegli un pool di dispositivi

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

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

Esempio:

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

La risposta include i pool di dispositivi Device Farm integrati, ad 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. Scegliere un pool di dispositivi e prendere nota del suo ARN.

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

Passaggio 3: carica il file della tua candidatura

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

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

Per ulteriori informazioni, consulta [create-upload](#).

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

Questo esempio crea un caricamento per un'applicazione Android:

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

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

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

```
}
```

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

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

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

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

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

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

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

Passaggio 4: carica il pacchetto di script di test

Quindi, caricare il pacchetto degli script di test.

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

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

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

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

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

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

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

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

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

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

```
{  
  "upload": {  
    "status": "SUCCEEDED",  
    "name": "MyTests.zip",  
    "created": 1535738627.195,  

```

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

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

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

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

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

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

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

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

```
}
```

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

Esempio:

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

La risposta contiene un URL prefirmato dove è possibile scaricare la specifica di test predefinita.

4. Questo esempio utilizza curl per scaricare la specifica di test predefinita e salvarla come `MyTestSpec.yml`:

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

5. Puoi modificare la specifica di test predefinita per soddisfare i requisiti di test e utilizzare la specifica di test modificata in sessioni di test future. Saltare questa fase per utilizzare la specifica di test predefinita così come avviene in un ambiente di test personalizzato.
6. Per creare un caricamento della specifica di test personalizzata, eseguire create-upload specificando il nome della specifica di test, il tipo della specifica di test e l'ARN di progetto.

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

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

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

```
{  
  "upload": {  
    "status": "INITIALIZED",  
    "category": "PRIVATE",  
    "name": "MyTestSpec.yml",  
    "created": 1535751101.221,  
    "url": "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/  
ExampleURL",
```

```

    "type": "APPIUM_JAVA_TESTNG_TEST_SPEC",
    "arn": "arn:aws:devicefarm:us-west-2:123456789101:upload:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE"
  }
}

```

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

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

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

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

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

```

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

```

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

Passaggio 6: Pianifica un'esecuzione di test

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

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

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

Per pianificare una sessione in un ambiente di test standard

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

Esempio:

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

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

```
{  
  "run": {  
    "status": "SCHEDULING",  
    "appUpload": "arn:aws:devicefarm:us-west-2:123456789101:upload:5e01a8c7-  
c861-4c0a-b1d5-12345appEXAMPLE",  
    "name": "MyTestRun",  
    "radios": {  
      "gps": true,  
      "wifi": true,  
      "nfc": true,  
      "bluetooth": true  
    },  
    "created": 1535756712.946,  
    "totalJobs": 179,  
    "completedJobs": 0,  
    "platform": "ANDROID_APP",  
    "result": "PENDING",  
  }  
}
```

```

    "devicePoolArn": "arn:aws:devicefarm:us-
west-2:123456789101:devicepool:5e01a8c7-c861-4c0a-b1d5-12345devicepoolEXAMPLE",
    "jobTimeoutMinutes": 150,
    "billingMethod": "METERED",
    "type": "APPIUM_JAVA_TESTNG",
    "testSpecArn": "arn:aws:devicefarm:us-west-2:123456789101:upload:5e01a8c7-
c861-4c0a-b1d5-12345specEXAMPLE",
    "arn": "arn:aws:devicefarm:us-west-2:123456789101:run:5e01a8c7-c861-4c0a-
b1d5-12345runEXAMPLE",
    "counters": {
        "skipped": 0,
        "warned": 0,
        "failed": 0,
        "stopped": 0,
        "passed": 0,
        "errored": 0,
        "total": 0
    }
}
}

```

Per ulteriori informazioni, consulta [schedule-run](#).

Per pianificare una sessione in un ambiente di test personalizzato

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

Esempio:

```

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

```

Per verificare lo stato della sessione di test

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

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

Per ulteriori informazioni, consulta [get-run](#). Per informazioni sull'utilizzo di Device Farm con AWS CLI, vedere [AWS CLI riferimento](#).

Creare un'esecuzione di test (API)

I passaggi sono gli stessi descritti nella AWS CLI sezione. Consultare [Crea un test run \(AWS CLI\)](#).

Queste informazioni sono necessarie per chiamare l'API [ScheduleRun](#):

- Un ARN di progetto. Consulta [Crea un progetto \(API\)](#) e [CreateProject](#).
- Un ARN di caricamento dell'applicazione. Consultare [CreateUpload](#).
- Un ARN di caricamento del pacchetto di test. Consultare [CreateUpload](#).
- Un ARN del pool di dispositivi. Consulta [Creazione di un pool di dispositivi](#) e [CreateDevicePool](#).

Note

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

Per informazioni sull'utilizzo dell'API Device Farm, 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 [Report in AWS Device Farm](#).

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

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

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

Important

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

Se l'esecuzione supera il timeout, l'esecuzione su quel dispositivo è arrestata forzatamente. Sono disponibili i risultati parziali, laddove possibile. Se si utilizza l'opzione di fatturazione con misurazione, l'esecuzione è fatturata fino a quel punto. Per ulteriori informazioni sui prezzi, consulta la 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 [Creazione di un progetto in AWS Device Farm](#), poi torna a questa pagina.

Imposta il timeout di esecuzione per un progetto

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

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

Imposta il timeout di esecuzione per un'esecuzione di test

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

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

Puoi utilizzare il network shaping per simulare le connessioni e le condizioni di rete durante il test delle tue app Android, iOS, 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 Wi-Fi 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 [Creazione di un progetto in AWS Device Farm](#).

2. Scegli l'applicazione, quindi scegli Avanti.
3. Configura il test, quindi scegli Avanti.
4. Seleziona i tuoi dispositivi, quindi scegli Avanti.
5. Nella sezione Impostazioni di posizione e rete, scegli un profilo di rete o scegli Crea profilo di rete per crearne uno personalizzato.

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

Crea un profilo di rete

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

1. Scegli Crea profilo di rete.

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.

Cancel Create

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

3. Scegli Create (Crea).

4. Termina di creare la tua sessione di test e avvia l'esecuzione.

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

General

Device pools

Network profiles

Uploads

Network profiles

Edit

Delete

Create network profile

	Name	Bandwidth (bps)	Delay (ms)	Jitter (ms)	Loss (%)	Description
		104857600 1048576	0 0	0 0	0 0	-
		104857600 1048576	0 0	0 0	0 0	-
		104857600 1048576	0 0	0 0	0 0	-

Modifica le condizioni della rete durante il test

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

Interruzione di un'esecuzione in AWS Device Farm

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

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

Argomenti

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

Interrompi una corsa (console)

1. Accedere alla console Device Farm all'indirizzo <https://console.aws.amazon.com/devicefarm>.
2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.
3. 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

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

Unique problems

Screenshots

Parsing result

Devices

Find device by status, device name, or OS

<

1

>

⌕

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

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

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

Verrà visualizzato un output simile al seguente:

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

```

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

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

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

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

Interrompere un'esecuzione (API)

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

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

Visualizzazione di un elenco di esecuzioni in AWS Device Farm

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

Argomenti

- [Visualizza un elenco di esecuzioni \(console\)](#)
- [Visualizza un elenco di esecuzioni \(AWS CLI\)](#)
- [Visualizzare un elenco di esecuzioni \(API\)](#)

Visualizza un elenco di esecuzioni (console)

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

Tip

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

Visualizza un elenco di esecuzioni (AWS CLI)

- Esegui il comando [list-runs](#).

Per visualizzare informazioni su una singola esecuzione, esegui il comando [get-run](#).

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

Visualizzare un elenco di esecuzioni (API)

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

Per visualizzare informazioni su una singola esecuzione, chiama l'API [GetRun](#).

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

Creazione di un pool di dispositivi in AWS Device Farm

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

Argomenti

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

Crea un pool di dispositivi (console)

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

- b. Scegli Aggiungi una regola.
- c. Per Campo (primo elenco a discesa), scegli una delle seguenti opzioni:
 - Per includere i dispositivi in base al nome del produttore, scegli Produttore del dispositivo.
 - Per includere i dispositivi in base al fattore di forma (tablet o telefono), scegli Fattore di forma.
 - Per includere i dispositivi in base al loro stato di disponibilità in base al carico, scegli Disponibilità.
 - Per includere solo dispositivi pubblici o privati, scegli Fleet Type.
 - Per includere i dispositivi in base al loro sistema operativo, scegli Piattaforma.
 - Alcuni dispositivi hanno un'etichetta o una descrizione aggiuntiva sul dispositivo. Puoi trovare i dispositivi in base al contenuto delle etichette selezionando le etichette delle istanze.
 - Per includere i dispositivi in base alla versione del sistema operativo, scegli Versione del sistema operativo.
 - Per includere i dispositivi in base al modello, scegli Modello.
- d. Per Operatore (secondo elenco a discesa), scegliete un'operazione logica (EQUALS, CONTAINS, ecc.) per includere i dispositivi in base alla query. Ad esempio, puoi scegliere di includere *Availability EQUALS AVAILABLE* i dispositivi che attualmente hanno lo stato. Available
- e. In Valore (terzo elenco a discesa), inserisci o scegli il valore che desideri specificare per i valori Campo e Operatore. I valori sono limitati in base alla scelta del campo. Ad esempio, se scegli Platform for Field, le uniche selezioni disponibili sono ANDROID e IOS. Allo stesso modo, se scegli Form Factor for Field, le uniche selezioni disponibili sono TELEFONO e TABLET.
- f. Per aggiungere un'altra regola, scegli Aggiungi una regola.

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

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

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

- h. Per eliminare una regola, scegli Rimuovi regola.
5. Se desideri includere o escludere manualmente singoli dispositivi, procedi come segue:
 - a. Scegli Crea pool di dispositivi statici.
 - b. Seleziona o deseleziona la casella accanto a ciascun dispositivo. È possibile selezionare o deselezionare le caselle solo se non si dispone di tutte le regole specificate.
6. Se si desidera includere o escludere tutti i dispositivi visualizzati, selezionare o deselezionare la casella nella riga di intestazione della colonna dell'elenco. Se desideri visualizzare solo le istanze private del dispositivo, scegli Visualizza solo le istanze private del dispositivo.

 Important

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

7. Scegli Create (Crea).

Crea un pool di dispositivi (AWS CLI)

 Tip

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

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

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

Creare un pool di dispositivi (API)

Tip

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

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

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

Analisi dei risultati dei test in AWS Device Farm

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

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

Argomenti

- [Visualizzazione dei report dei test in Device Farm](#)
- [Scaricamento di artefatti in Device Farm](#)

Visualizzazione dei report dei test in Device Farm

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

Argomenti

- [Prerequisiti](#)

- [Visualizza i report](#)
- [Stati dei risultati del test Device Farm](#)

Prerequisiti

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

1. Per creare un'esecuzione, consulta [Creazione di un'esecuzione di test in Device Farm](#) e torna a questa pagina.
2. Verifica che l'esecuzione sia completa. Durante l'esecuzione del test, la console Device Farm visualizza un'icona in sospeso



per le esecuzioni in corso. Ogni dispositivo in esecuzione inizierà anche con l'icona in sospeso, quindi passerà



all'inizio del test. Al termine di ogni test, accanto al nome del dispositivo viene visualizzata l'icona del risultato del test. Una volta completati tutti i test, l'icona in sospeso accanto all'esecuzione diventa l'icona del risultato del test. Per ulteriori informazioni, consulta [Stati dei risultati del test Device Farm](#).

icon

Visualizza i report

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

Argomenti

- [Visualizza la pagina di riepilogo dell'esecuzione del test](#)
- [Visualizza segnalazioni di problemi uniche](#)
- [Visualizza i report sui dispositivi](#)
- [Visualizza i report della suite di test](#)
- [Visualizzazione dei report di test](#)
- [Visualizza 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>.
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

Q

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 [Stati di un singolo test](#).

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

La scheda Performance 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 [Stati per test multipli](#).

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 [Stati per test multipli](#).

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 [Stati di un singolo test](#).

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

La scheda Performance 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 grafico della CPU mostra la percentuale di CPU utilizzata dall'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 un **MM:SS.SSS** formato, dove **M** rappresenta i minuti e **S** rappresenta i secondi.
- PID indica l'identificatore del processo (PID) che ha creato la voce di log. Tutte le voci di log create da un'app su un dispositivo hanno lo stesso PID.
- Level (Livello) indica il livello registrato per la voce di log. Ad esempio, `Logger.debug("This is a message!")` registra il livello Debug. I valori possibili sono:
 - Avviso
 - Critico
 - Esegui il debug
 - Emergenza
 - Errore
 - Errore
 - 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.

Stati dei risultati del test Device Farm

La console Device Farm mostra icone che consentono di valutare rapidamente lo stato dell'esecuzione del test completata. Per ulteriori informazioni sui test in Device Farm, vedere [Report in AWS Device Farm](#).

Argomenti

- [Stati di un singolo test](#)
- [Stati per test multipli](#)

Stati di un singolo test

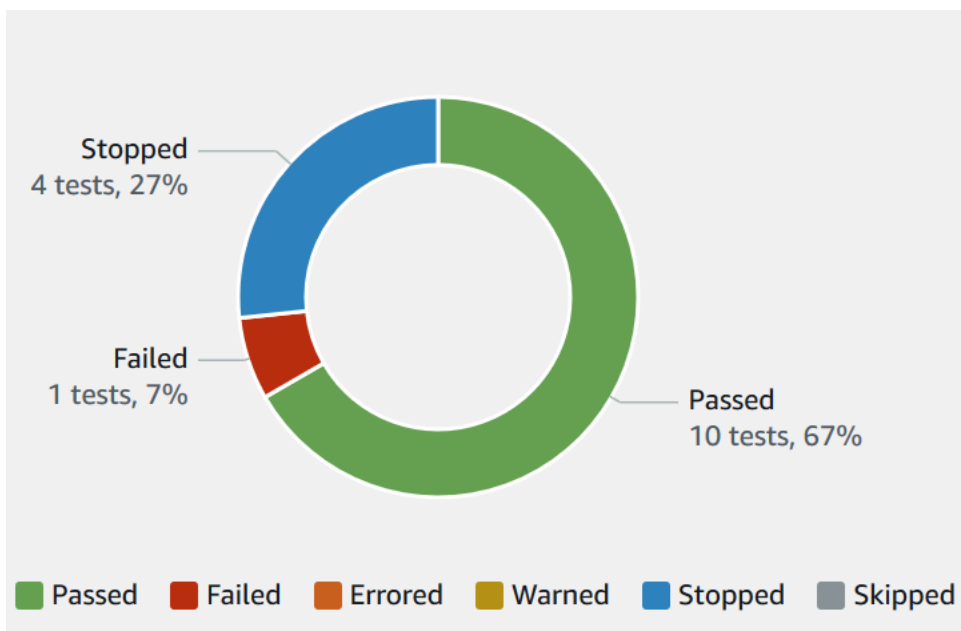
Per i report che descrivono un singolo test, Device Farm visualizza un'icona che rappresenta lo stato del risultato del test:

Descrizione	Icon
Il test ha avuto esito positivo.	

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

Stati per test multipli

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



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

I grafici sono sempre codificati a colori ed etichettati.

Scaricamento di artefatti in Device Farm

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

Puoi scaricare artefatti creati durante la sessione di test:

File

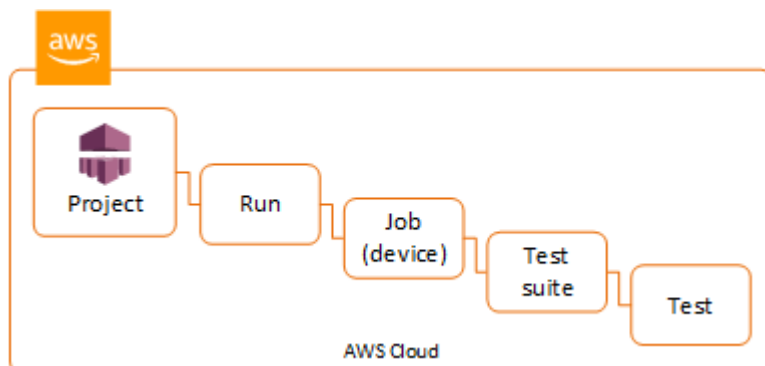
File generati durante l'esecuzione del test, inclusi i report di Device Farm. Per ulteriori informazioni, consulta [Visualizzazione dei report dei test in Device Farm](#).

Log

Output da ciascun test nella sessione.

Screenshot

Immagini di schermata registrate per ogni test nella sessione.



Scarica artefatti (console)

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

Per ulteriori informazioni su come scaricare gli artefatti in un ambiente di test personalizzato, consulta [Scaricamento di artefatti in un ambiente di test personalizzato](#).

Scarica gli artefatti ()AWS CLI

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

Argomenti

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

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

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

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

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

Esempio:

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

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

```
{  
  "tests": [  

```

```

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

```

Fase 2: Elenca i tuoi artefatti

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

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

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

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

La risposta contiene un URL di download per ogni artefatto.

```

{
  "artifacts": [

```

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

Passaggio 3: scarica i tuoi artefatti

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

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

Scarica gli artefatti (API)

Il [ListArtifacts](#) metodo API Device Farm restituisce un elenco di elementi, come file, schermate e registri. Ogni artefatto dispone di un URL in modo che tu possa scaricare il file.

Scaricamento di artefatti in un ambiente di test personalizzato

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

Puoi scaricare questi artefatti creati durante la sessione di test:

Output di specifica di test

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

Artefatti personalizzati

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

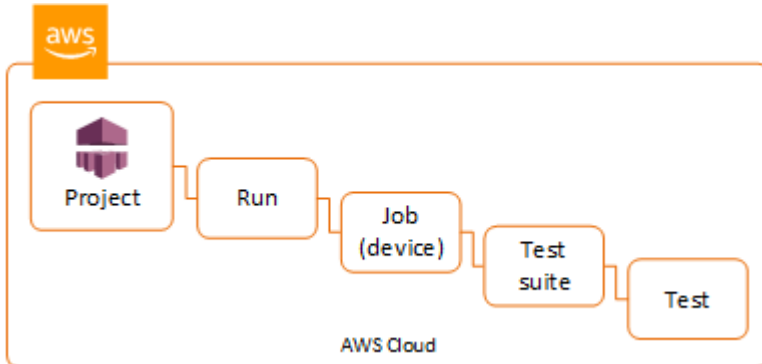
Script della shell di specifica di test

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

File di specifica di test

Il file YAML utilizzato nella sessione di test.

Per ulteriori informazioni, consulta [Scaricamento di artefatti in Device Farm](#).



Etichettatura delle risorse di AWS Device Farm

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

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

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

Applicazione di tag alle risorse

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

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

```
import boto3

client = boto3.client('resourcegroupstaggingapi')

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

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

Ricerca di risorse per tag

Per eseguire la ricerca delle risorse in base ai tag, utilizzare l'operazione `GetResources` dall'endpoint `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 (`devicefarm:testgrid-session`risorse) di test del browser desktop Device Farm con il tag `stack` che ha il valore `reproduction`:


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

Rimozione dei tag dalle risorse

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

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

Framework di test e test integrati in AWS Device Farm

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

Per ulteriori informazioni su come Device Farm esegue i test, vedere [Ambienti di test in AWS Device Farm](#).

Framework di test

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

Framework di test delle applicazioni Android

- [Appium](#)
- [Instrumentation](#)

Framework di test delle applicazioni iOS

- [Appium](#)
- [XCTest](#)
- [XCTest UI](#)

Framework di test delle applicazioni Web

Le applicazioni Web sono supportate utilizzando Appium. Per ulteriori informazioni su come trasferire i test su Appium, consulta [Test 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 [Ambienti di test personalizzati in AWS Device Farm](#).

Supporto per la versione Appium

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

Tipi di test integrati

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

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

Test 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 i collegamenti ai test di lavoro, consulta [Device Farm Sample App per Android](#) e [Device Farm Sample App per iOS](#) su GitHub.

Per ulteriori informazioni sui test in Device Farm, vedere [Framework di test e test integrati in AWS Device Farm](#).

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 YAML delle specifiche di test:

```
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 YAML della specifica di test:

```
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 YAML della specifica di test:

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

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

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

Integrazione dei test Appium con Device Farm

Utilizza le seguenti istruzioni per integrare i test Appium con AWS Device Farm. Per ulteriori informazioni sull'utilizzo dei test Appium in Device Farm, vedere. [Test Appium e AWS Device Farm](#)

Configura il tuo pacchetto di test Appium

Utilizza le seguenti istruzioni per configurare il pacchetto del test.

Java (JUnit)

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

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

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

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

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

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

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

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

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

Il seguente codice XML è una definizione di assembly che, una volta configurata, indica a Maven di creare un file.zip che contenga tutto ciò che si trova nella radice della directory di output della build e della directory: `dependency-jars`

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

5. Modifica `pom.xml` per utilizzarlo per `maven-assembly-plugin` impacchettare i test e tutte le dipendenze in un unico file.zip.

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

```
<plugin>
  <artifactId>maven-assembly-plugin</artifactId>
```

```
<version>2.5.4</version>
<executions>
  <execution>
    <phase>package</phase>
    <goals>
      <goal>single</goal>
    </goals>
    <configuration>
      <finalName>zip-with-dependencies</finalName>
      <appendAssemblyId>false</appendAssemblyId>
      <descriptors>
        <descriptor>src/main/assembly/zip.xml</descriptor>
      </descriptors>
    </configuration>
  </execution>
</executions>
</plugin>
```

Note

Se ricevi un messaggio di errore indicante che l'annotazione non è supportata nella versione 1.3, aggiungi quanto segue a `pom.xml`:

```
<plugin>
  <artifactId>maven-compiler-plugin</artifactId>
  <configuration>
    <source>1.7</source>
    <target>1.7</target>
  </configuration>
</plugin>
```

Java (TestNG)

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

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

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

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

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

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

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

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

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

Il seguente codice XML è una definizione di assembly che, una volta configurata, indica a Maven di creare un file.zip che contenga tutto ciò che si trova nella radice della directory di output della build e della directory: `dependency-jars`

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

5. Modifica `pom.xml` per utilizzarlo per `maven-assembly-plugin` impacchettare i test e tutte le dipendenze in un unico file.zip.

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

```
<plugin>
  <artifactId>maven-assembly-plugin</artifactId>
```

```
<version>2.5.4</version>
<executions>
  <execution>
    <phase>package</phase>
    <goals>
      <goal>single</goal>
    </goals>
    <configuration>
      <finalName>zip-with-dependencies</finalName>
      <appendAssemblyId>false</appendAssemblyId>
      <descriptors>
        <descriptor>src/main/assembly/zip.xml</descriptor>
      </descriptors>
    </configuration>
  </execution>
</executions>
</plugin>
```

Note

Se ricevi un messaggio di errore indicante che l'annotazione non è supportata nella versione 1.3, aggiungi quanto segue a `pom.xml`:

```
<plugin>
  <artifactId>maven-compiler-plugin</artifactId>
  <configuration>
    <source>1.7</source>
    <target>1.7</target>
  </configuration>
</plugin>
```

Node.JS

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

- [Node Version Manager \(nvm\)](#)

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

- Node.js
- npm-bundle (installato a livello globale)

1. Verificare che nvm sia presente

```
command -v nvm
```

Dovresti vedere nvm come output.

Per ulteriori informazioni, vedere [nvm](#) on. GitHub

2. Eseguire questo comando per installare Node.js:

```
nvm install node
```

È possibile specificare una determinata versione di Node.js:

```
nvm install 11.4.0
```

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

```
node -v
```

4. Installare npm-bundle globalmente:

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

Python

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

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

 Tip

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

2. Installare `py.test` nell'ambiente virtuale.

```
$ pip install pytest
```

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

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

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

```
$ find tests/
```

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

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

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

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

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

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

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

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

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

Ruby

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

- [Ruby Version Manager \(RVM\)](#)

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

- Ruby
 - Bundler (viene in genere installato con Ruby).
1. Installare le chiavi richieste, RVM e Ruby. Per istruzioni, consulta [Installazione di RVM](#) sul sito Web relativo.

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

Note

RVM è caricato come funzione solo per shell bash.

2. Verifica che rvm sia installato correttamente.

```
command -v rvm
```

Dovresti vedere rvm come output.

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

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

Verificare di utilizzare la versione richiesta di Ruby:

```
ruby -v
```

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

```
bundle config specific_platform true
```

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

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

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

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

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

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

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

Crea un file di pacchetto di test compresso

Warning

In Device Farm, la struttura delle cartelle dei file nel pacchetto di test compresso è importante e alcuni strumenti di archiviazione modificheranno implicitamente la struttura del file ZIP.

Ti consigliamo di seguire le utilità della riga di comando specificate di seguito anziché utilizzare le utilità di archiviazione integrate nel file manager del desktop locale (come Finder o Windows Explorer).

Ora, raggruppare i test per Device Farm.

Java (JUnit)

Creare i test:

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

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

Java (TestNG)

Creare i test:

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

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

Node.JS

1. Verifica il progetto.

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

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

```
npm install
```

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

Note

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

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

```
npm-bundle
```

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

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

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

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

Python

Python 2

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

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

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

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

Python 3

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

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

Ruby

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

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

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

```
rvm gemset use myGemset
```

3. Consulta il codice sorgente.

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

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

```
bundle install
```

 Note

A questo punto dovresti essere in grado di eseguire i test in locale. Utilizzare questo comando per eseguire un test in locale:

```
bundle exec $test_command
```

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

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

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

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

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

Carica il tuo pacchetto di test su Device Farm

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

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

Se hai già un progetto, puoi sceglierlo per caricarvi i test.

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

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

 Note

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

Per i test delle applicazioni Web per dispositivi mobili

In Impostazioni di esecuzione, scegli App Web.

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

 Note

Device Farm non modifica i test Appium.

Acquisisci schermate dei tuoi test (opzionale)

Durante i test, è possibile acquisire screenshot.

Device Farm imposta la proprietà `DEVICEFARM_SCREENSHOT_PATH` su percorso pienamente qualificato sul sistema locale di file in cui Device Farm si aspetta il salvataggio degli screenshot Appium. La directory specifica del test in cui sono archiviati gli screenshot è definita in fase di

esecuzione. Gli screenshot vengono recuperati automaticamente nei report Device Farm. Per visualizzare gli screenshot, nella console Device Farm, selezionare la sezione Screenshots (Screenshot).

Per ulteriori informazioni sull'acquisizione di screenshot nei test Appium, vedere [Acquisizione di screenshot](#) nella documentazione dell'API Appium.

Test Android in AWS Device Farm

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

Per ulteriori informazioni sui test in Device Farm, vedere [Framework di test e test integrati in AWS Device Farm](#).

Framework di test delle applicazioni Android

Sono disponibili i seguenti test per i dispositivi Android.

- [Appium](#)
- [Instrumentation](#)

Tipi di test integrati per Android

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

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

Strumentazione per Android e AWS Device Farm

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

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

Per ulteriori informazioni sui test in Device Farm, vedere [Framework di test e test integrati in AWS Device Farm](#).

Argomenti

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

Che cos'è la strumentazione?

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

Considerazioni per i test di strumentazione Android

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

Verifica la compatibilità del sistema operativo Android

Consulta la [documentazione di Android](#) per assicurarti che Instrumentation sia compatibile con la versione del tuo sistema operativo Android.

Esecuzione dalla riga di comando

Per eseguire i test di Instrumentation dalla riga di comando, segui la documentazione di [Android](#).

System Animations (Animazioni di sistema)

Secondo la [documentazione Android per i test di Espresso](#), si consiglia di disattivare le animazioni di sistema durante i test su dispositivi reali. Device Farm disattiva automaticamente le impostazioni Window Animation Scale, Transition Animation Scale e Animator Duration Scale quando viene eseguito con il test runner di strumentazione [JUnit4support.test.runner.Android](#) Runner.

Test Recorders (Registratori di test)

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

Analisi dei test in modalità standard

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

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

Per testarlo in un ambiente locale:

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

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

Integrazione della strumentazione Android con Device Farm

Note

Utilizza le seguenti istruzioni per integrare i test della strumentazione Android con AWS Device Farm. Per ulteriori informazioni sull'utilizzo dei test di strumentazione in Device Farm, vedere. [Strumentazione per Android e AWS Device Farm](#)

Carica i tuoi test di strumentazione Android

Usa la console Device Farm per caricare i test.

1. Accedere alla console Device Farm all'indirizzo <https://console.aws.amazon.com/devicefarm>.
2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.
3. Nell'elenco dei progetti, scegli il progetto su cui 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 [Creazione di un progetto in AWS Device Farm](#).

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

(Facoltativo) Scatta schermate nei test della strumentazione Android

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

Per acquisire screenshot, chiamare uno dei seguenti metodi:

- Per Robotium, chiamare il metodo `takeScreenShot` (ad esempio, solo `.takeScreenShot()`).
- Per Spoon, chiamare il metodo `screenshot`, ad esempio:

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

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

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

Test iOS in AWS Device Farm

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

Per ulteriori informazioni sui test in Device Farm, vedere [Framework di test e test integrati in AWS Device Farm](#).

Framework di test delle applicazioni iOS

Sono disponibili i seguenti test per i dispositivi iOS.

- [Appium](#)
- [XCTest](#)
- [XCTest UI](#)

Tipi di test integrati per iOS

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

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

Integrazione di Device Farm con XCTest iOS

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

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

Per ulteriori informazioni sui test in Device Farm, vedere [Framework di test e test integrati in AWS Device Farm](#).

Argomenti

- [Crea i pacchetti per la tua XCTest corsa](#)
- [Carica i pacchetti per la tua XCTest corsa su Device Farm](#)

Crea i pacchetti per la tua XCTest corsa

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

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

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

Per generare l'output di compilazione per l'app

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

Per creare il pacchetto di app

1. Nel navigatore di progetto in Xcode, aprire sotto Products (Prodotti) il menu contestuale per il file denominato `app-project-name.app`. Quindi, scegliere Show in Finder (Mostra in Finder). Finder apre una cartella denominata Debug-iphoneros, 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-iphoneros il menu contestuale per il file **app-project-name**.app. Quindi, scegliere Show Package Contents (Mostra contenuti pacchetto).
2. Nei contenuti del pacchetto, aprire la cartella Plugins. Questa cartella contiene un file denominato **app-project-name**.xctest.
3. Aprire il menu contestuale per questo file e scegliere Compress "**app-project-name.xctest**" (Comprimi "app-project-name.xctest"). Viene creato un file denominato **app-project-name**.xctest.zip.

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

Carica i pacchetti per la tua XCTest corsa su Device Farm

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

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

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

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

Note

Il pacchetto .ipa deve essere compilato per il test.

8. In Configura test, nella sezione Seleziona framework di test, scegli XCTest. Quindi, seleziona Scegli file in Carica app.

9. Cerca il .zip file che contiene il XCTest pacchetto per la tua app e caricalo.
10. Completare le fasi rimanenti del processo di creazione del progetto. Sarà quindi possibile selezionare i dispositivi che si desidera testare e specificare lo stato del dispositivo.
11. Scegli Crea esegui. Device Farm esegue il test e mostra i risultati nella console.

Integrazione dell' XCTest interfaccia utente per iOS con Device Farm

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

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

Per informazioni generali sui test in Device Farm, vedere [Framework di test e test integrati in AWS Device Farm](#).

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

Argomenti

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

Prepara i test XCTest dell'interfaccia utente iOS

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

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

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

Opzione 1: creazione di un pacchetto XCTest UI .ipa

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

Per creare un file .ipa:

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

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

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

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

Device Farm genera automaticamente un .xctestrun file per l'esecuzione della suite di test XCTest dell'interfaccia utente completa. Se desideri utilizzare il tuo .xctestrun file su Device Farm, puoi comprimere .xctestrun i file e la directory dell'app in un .zip file. Se hai già un .ipa file per il tuo pacchetto di test, puoi includerlo qui invece di **-Runner.app*.

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

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

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

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

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

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

Note

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

Carica i test XCTest dell'interfaccia utente iOS

Usa la console Device Farm per caricare i test.

1. Accedere alla console Device Farm all'indirizzo <https://console.aws.amazon.com/devicefarm>.
2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.
3. Nell'elenco dei progetti, scegli il progetto su cui 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 [Creazione di un progetto in AWS Device Farm](#)

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

 Note

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

8. In Configura test, nella sezione Seleziona framework di test, scegli XCTest UI. Quindi, seleziona Scegli file in Carica app.
9. Cerca e scegli il file.ipa o.zip che contiene il tuo test runner XCTest dell'interfaccia utente iOS.
10. Completa i passaggi rimanenti del processo di creazione dell'esecuzione. Selezionerai i dispositivi su cui desideri eseguire il test e, facoltativamente, specificherai una configurazione aggiuntiva.
11. Scegli Crea esegui. Device Farm esegue il test e mostra i risultati nella console.

Test di app Web in AWS Device Farm

Device Farm fornisce test con Appium per applicazioni web. Per ulteriori informazioni sulla configurazione dei test Appium su Device Farm, consulta [the section called "Appium"](#)

Per ulteriori informazioni sui test in Device Farm, vedere [Framework di test e test integrati in AWS Device Farm](#).

Regole per i dispositivi misurati e non misurati

Per misurazione si intende la fatturazione per i dispositivi. Per impostazione predefinita, i dispositivi Device Farm vengono misurati e ti viene addebitato un addebito al minuto una volta esauriti i minuti

di prova gratuiti. Puoi anche scegliere di acquistare dispositivi non misurati, per effettuare test illimitatamente a un costo mensile fisso. Per ulteriori informazioni sui prezzi, consulta la pagina dei [prezzi di AWS Device Farm](#).

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

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

Test integrati in AWS Device Farm

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

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

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

Per ulteriori informazioni sui test e sui framework di test in Device Farm, vedere [Framework di test e test integrati in AWS Device Farm](#)

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

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

Per ulteriori informazioni sui test in Device Farm, vedere [Framework di test e test integrati in AWS Device Farm](#).

Per eseguire il fuzz test integrato

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

2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.
3. Nell'elenco dei progetti, scegli il progetto in cui desideri eseguire il fuzz test integrato.

 Tip

Puoi usare la barra di ricerca per filtrare l'elenco dei progetti per nome.

Per creare un progetto, segui le istruzioni indicate nella sezione [Creazione di un progetto in AWS Device Farm](#).

4. Scegli Crea esegui.
5. In Impostazioni di esecuzione, seleziona il tipo di corsa nella sezione Tipo di corsa. Seleziona l'app per Android se non hai un'app pronta per il test o se stai testando un'app per Android (.apk). Seleziona App iOS se stai testando un'app iOS (.ipa).
6. In Seleziona app, scegli Seleziona app di esempio fornita da Device Farm se non hai un'app disponibile per il test. Se porti la tua app, seleziona Carica la tua app e scegli il file dell'applicazione.
7. In Configura test, nella sezione Seleziona framework di test, scegli Built-in: Fuzz.
8. Se compare una delle impostazioni seguenti, è possibile accettare i valori predefiniti o specificare i propri:
 - Conteggio eventi: specificare un numero compreso tra 1 e 10.000, che rappresenta il numero di eventi di interfaccia utente che il test Fuzz deve eseguire.
 - Event throttle: specifica un numero compreso tra 0 e 1.000, che rappresenta il numero di millisecondi di attesa del fuzz test prima di eseguire il successivo evento dell'interfaccia utente.
 - Seed randomizer: specificare un numero che il test Fuzz utilizzerà per creare casualmente eventi di interfaccia utente. Se si specifica lo stesso numero per test Fuzz successivi, le sequenze di eventi saranno identiche.
9. Completa le istruzioni rimanenti per selezionare i dispositivi e avviare l'esecuzione.

Ambienti di test personalizzati in AWS Device Farm

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

I vantaggi della modalità personalizzata rispetto alla modalità standard includono:

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

Per ulteriori informazioni, consulta [Caricamento di una specifica di test personalizzata utilizzando AWS CLI](#) e [Creazione di un'esecuzione di test in Device Farm](#).

Argomenti

- [Sintassi delle specifiche di test in Device Farm](#)
- [Esempio del file delle specifiche di test di Device Farm](#)
- [Ambiente di test Amazon Linux 2 per test Android](#)

- [Variabili di ambiente in Device Farm](#)
- [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 in Device Farm

La specifica di test è un file che usi per definire ambienti di test personalizzati in AWS Device Farm. Per ulteriori informazioni sugli ambienti personalizzati e sul file delle specifiche di test, consulta.

[Ambienti di test personalizzati in AWS Device Farm](#)

Di seguito è riportata la struttura del file delle specifiche di test YAML. Di seguito la struttura è riportata una descrizione di ciascuna proprietà.

Per vedere un esempio di file di specifiche di test, fare riferimento a [Esempio del file delle specifiche di test di Device Farm](#).

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

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

 Important

Una specifica di test deve essere formattata come file YAML valido. Se l'indentazione o la spaziatura della tua specifica di test non è valida, il tuo test può non riuscire. Le schede non sono consentite nei file YAML. Puoi usare un validatore YAML per verificare se la tua specifica di test è un file YAML valido. Per ulteriori informazioni, consulta il [sito Web YAML](#).

Esempio del file delle specifiche di test di Device Farm

La specifica di test è un file che usi per definire ambienti di test personalizzati in AWS Device Farm. Per ulteriori informazioni sugli ambienti personalizzati e sul file delle specifiche di test, consulta.. [Ambienti di test personalizzati in AWS Device Farm](#) Per ulteriori informazioni sulla struttura e sul contenuto del file delle specifiche di test, vedere. [Sintassi delle specifiche di test in Device Farm](#)

Di seguito è riportato un esempio di una specifica di test YAML di Device Farm che configura l'esecuzione di un 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
```

Ambiente di test Amazon Linux 2 per test Android

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

L'host 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

- [Librerie software preinstallate per supportare i test Device Farm dei dispositivi Android](#)
- [Intervalli IP supportati per l'ambiente di test Amazon Linux 2 in Device Farm](#)
- [Utilizzo dello `devicefarm-cli` strumento in AWS Device Farm](#)
- [Selezione dell'host di test Android da utilizzare in Device Farm](#)

- [Esempio: file delle specifiche di test di Device Farm che mostra un host di test Android](#)
- [Migrazione all'Amazon Linux 2 Test Host in AWS Device Farm](#)

Librerie software preinstallate per supportare i test Device Farm dei dispositivi Android

AWS Device Farm utilizza macchine host Amazon Elastic Compute Cloud (EC2) che eseguono Amazon Linux 2 per eseguire test Android. 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 tuo pacchetto di test, scaricarlo da Internet o accedere a fonti private all'interno del tuo VPC (vedi [VPC ENI](#) per ulteriori informazioni). Per ulteriori informazioni, consulta 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 per l'ambiente di test Amazon Linux 2 in Device Farm

I clienti spesso hanno bisogno di conoscere l'intervallo di IP da cui proviene il traffico di Device Farm, in particolare per configurare i firewall e le impostazioni di sicurezza. Per gli host 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 gateway NAT, limitando l'intervallo IP ai seguenti indirizzi:

Intervalli IP

44.236.137143

5213,151,44

5235,189,1991

54201.250,26

Per ulteriori informazioni sugli ambienti di test Android in Device Farm, consulta [Ambiente di test Amazon Linux 2 per test Android](#).

Utilizzo dello **devicefarm-cli** strumento in AWS Device Farm

AWS Device Farm utilizza macchine host Amazon Elastic Compute Cloud (EC2) che eseguono Amazon Linux 2 per eseguire test Android. 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 utilizzare lo strumento per modificare la versione di Python da **3.10** a **3.9** nel file delle specifiche di test, esegui i seguenti comandi:

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

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

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

Tip

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

Per ulteriori informazioni su come Device Farm testa i dispositivi Android, vedere [Ambiente di test Amazon Linux 2 per test Android](#).

Per ulteriori informazioni sul software preinstallato sull'host di test Amazon Linux 2, consulta [Librerie software preinstallate per supportare i test Device Farm dei dispositivi Android](#).

Selezione dell'host di test Android da utilizzare in Device Farm

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.

Tieni presente che Legacy Android Test Host supporta solo le versioni Android 14 e precedenti. Utilizza l'host `amazon_linux_2` per le versioni Android 15 e successive.

AWS Device Farm utilizza macchine host Amazon Elastic Compute Cloud (EC2) che eseguono Amazon Linux 2 per eseguire test Android. 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
```

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, usa `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: file delle specifiche di test di Device Farm che mostra un host di test Android

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
  # learn 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'Amazon Linux 2 Test Host in AWS Device Farm

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>.
2. Vai al progetto Device Farm contenente i tuoi test di automazione.
3. Scegli Crea ed esegui 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 nel PATH dell'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 file JAR preinstallato nel percorso di classe, mentre l'host precedente ne conteneva uno per il framework TestNG (tramite una variabile di ambiente). `$DEVICEFARM_TESTNG_JAR` Consigliamo ai clienti di impacchettare i file JAR necessari per i propri framework di test all'interno del pacchetto di test e di rimuovere le istanze della variabile dai file delle `$DEVICEFARM_TESTNG_JAR` specifiche di test. Per ulteriori informazioni, consulta [Working with Appium e 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 in Device Farm

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 in Device Farm](#)
- [Variabili di JUnit ambiente Java Appium in Device Farm](#)

- [Variabili di ambiente Appium Java TestNg in Device Farm](#)
- [XCUITest variabili di ambiente in Device Farm](#)

Variabili di ambiente comuni in Device Farm

Questa sezione descrive le variabili di ambiente comuni ai test della piattaforma Android e ai test della piattaforma iOS in AWS Device Farm. Per ulteriori informazioni sulle variabili di ambiente in Device Farm, vedere [Variabili di ambiente in Device Farm](#).

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 JUnit ambiente Java Appium in Device Farm

Questa sezione descrive le variabili di ambiente utilizzate dai JUnit test Java di Appium in un ambiente di test personalizzato in AWS Device Farm. Per ulteriori informazioni sulle variabili di ambiente in Device Farm, vedere [Variabili di ambiente in Device Farm](#).

\$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 in Device Farm

Questa sezione descrive le variabili di ambiente utilizzate dai test Appium Java TestNg in un ambiente di test personalizzato in Device Farm. Per ulteriori informazioni sulle variabili di ambiente in Device Farm, vedere [Variabili di ambiente in Device Farm](#).

\$DEVICEFARM_TESTNG_JAR

Il percorso al file .jar TestNG.

\$DEVICEFARM_TEST_PACKAGE_PATH

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

XCUITest variabili di ambiente in Device Farm

Questa sezione descrive le variabili di ambiente utilizzate dal XCUITest test in un ambiente di test personalizzato in Device Farm. Per ulteriori informazioni sulle variabili di ambiente in Device Farm, vedere [Variabili di ambiente in Device Farm](#).

\$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.

\$DEVICEFARM_XCTEST_BUILD_DIRECTORY

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

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

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

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

Per ulteriori informazioni sulle modalità standard e personalizzate in Device Farm, vedere [Ambienti di test in AWS Device Farm](#) e [Ambienti di test personalizzati in AWS Device Farm](#).

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 lo sarebbero se fossero eseguiti sui computer locali. I risultati

dei test sono gli stessi che si ottengono localmente e sono disponibili per il download al termine dell'esecuzione del lavoro.

2. Personalizzazione o flessibilità: la modalità di esecuzione standard analizza il pacchetto di test per rilevare il numero di test e quindi esegue ciascun test separatamente. Tieni presente che non è garantito che i test vengano eseguiti nell'ordine specificato. Di conseguenza, i test che richiedono una particolare sequenza di esecuzione potrebbero non funzionare come previsto. Inoltre, non è possibile personalizzare l'ambiente del computer host o trasmettere i file di configurazione che potrebbero essere necessari per eseguire i test in un determinato modo.

Al contrario, la modalità personalizzata consente di configurare l'ambiente del computer host, inclusa la possibilità di installare software aggiuntivo, applicare filtri ai test, passare file di configurazione e controllare la configurazione di esecuzione dei test. Ciò avviene tramite un file yaml (chiamato anche file testspec) che è possibile modificare aggiungendovi comandi di shell. Questo file yaml viene convertito in uno script di shell che viene eseguito sulla macchina host di test. Puoi salvare più file yaml e sceglierne uno dinamicamente in base alle tue esigenze quando pianifichi un'esecuzione.

3. Video e registrazione in diretta: sia la modalità di esecuzione standard che quella personalizzata forniscono video e registri per i test. Tuttavia, in modalità standard, il video e i registri predefiniti dei test vengono visualizzati solo dopo il completamento dei test.

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

Tip

Se il tuo caso d'uso 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. Accedi AWS Management Console e apri la console Device Farm all'indirizzo <https://console.aws.amazon.com/devicefarm/>.
2. Scegli il tuo progetto e poi inizia una nuova esecuzione di automazione.

3. Carica la tua app (o web app selezionala), scegli il tipo di framework di test, carica il pacchetto di test, quindi, sotto il Choose your execution environment parametro, scegli l'opzione Run your test in a custom environment.
4. Per impostazione predefinita, il file delle specifiche di test di esempio di Device Farm viene visualizzato e modificato. Questo file di esempio può essere utilizzato come punto di partenza per provare i test in [modalità ambiente personalizzato](#). Quindi, una volta verificato che i test funzionino correttamente dalla console, puoi modificare qualsiasi integrazione di API, CLI e pipeline con Device Farm per utilizzare questo file delle specifiche di test come parametro durante la pianificazione delle esecuzioni dei test. [Per informazioni su come aggiungere un file delle specifiche di test come parametro per le esecuzioni, consulta la sezione relativa ai testSpecArn parametri per l'API nella nostra guida all'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.

iOS XCUITest

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

Estensione degli ambienti di test personalizzati in Device Farm

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

Per ulteriori informazioni sugli ambienti di test personalizzati in Device Farm, vedere [Ambienti di test personalizzati in AWS Device Farm](#).

Argomenti

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

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

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

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

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

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

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

⚠ Warning

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

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

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

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

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

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

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

noReset e fullReset

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

ignoreUnimportantViews(Solo Android)

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

skipUnlock(Solo Android)

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

webDriverAgentUrl(solo iOS)

Indica ad Appium di presumere che una dipendenza iOS essenziale sia già in esecuzione e disponibile per accettare richieste HTTP all'URL specificato. `webdriverAgent` Se `webdriverAgent` non è già attivo e funzionante, Appium può impiegare del tempo all'inizio di una suite di test per avviare `webdriverAgent`. Se lo avvii `webdriverAgent` da solo e lo fai `http://localhost:8100` quando avvii Appium, puoi avviare la tua suite di test più velocemente. `webdriverAgentUrl` Nota che questa funzionalità non dovrebbe mai essere utilizzata insieme alla `useNewWDA` funzionalità.

È possibile utilizzare il codice seguente per iniziare `webdriverAgent` dal file delle specifiche di test sulla porta locale del 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](#).

Per ulteriori modi per estendere la suite di test e ottimizzare i test, consulta [Estensione degli ambienti di test personalizzati in Device Farm](#)

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

È possibile fare in modo che Device Farm chiami un webhook al termine dell'utilizzo curl di ogni suite di test. Il processo per eseguire questa operazione varia a seconda della destinazione e della formattazione. Per il tuo webhook specifico, consulta la documentazione relativa a quel webhook. L'esempio seguente pubblica un messaggio ogni volta che una suite di test termina su un webhook Slack:

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

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

Per ulteriori modi per estendere la suite di test e ottimizzare i test, consulta [Estensione degli ambienti di test personalizzati in Device Farm](#)

Non sei limitato a usare per curl chiamare i webhook. I pacchetti di test possono includere script e strumenti aggiuntivi, purché siano compatibili con l'ambiente di esecuzione Device Farm. Ad esempio, il pacchetto di test può includere script ausiliari che inoltrano richieste ad altri. APIs Assicurati che tutti i pacchetti richiesti siano installati insieme ai requisiti della tua suite di test. Per aggiungere uno script che viene eseguito dopo il completamento della suite di test, includi lo script nel pacchetto di test e aggiungi quanto segue alle specifiche di test:

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

Note

La manutenzione delle chiavi API o degli altri token di autenticazione utilizzati nel pacchetto di test è una tua responsabilità. Ti consigliamo di mantenere qualsiasi forma di credenziale di sicurezza al di fuori del controllo del codice sorgente, di utilizzare credenziali con il minor numero di privilegi possibile e di utilizzare token revocabili e di breve durata quando possibile.

Per verificare i requisiti di sicurezza, consulta la documentazione della terza parte che utilizzi. APIs

Se prevedi di utilizzare AWS i servizi come parte della tua suite di esecuzione dei test, dovresti utilizzare le credenziali temporanee IAM, generate al di fuori della tua suite di test e incluse nel pacchetto di test. Queste credenziali devono avere il minor numero di autorizzazioni concesse e la durata di vita più breve possibile. Per ulteriori informazioni sulla creazione di credenziali temporanee, consulta [Richiesta di credenziali di sicurezza temporanee nella Guida per l'utente IAM](#).

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

Aggiungere file aggiuntivi al pacchetto di test in Device Farm

Potresti voler utilizzare file aggiuntivi come parte dei tuoi test come file di configurazione aggiuntivi o dati di test aggiuntivi. Puoi aggiungere questi file aggiuntivi al tuo pacchetto di test prima di caricarlo AWS Device Farm, quindi accedervi dalla modalità ambiente personalizzata. Fondamentalmente, tutti i formati di caricamento dei pacchetti di test (ZIP, IPA, APK, JAR, ecc.) sono formati di archivio dei pacchetti che supportano le operazioni ZIP standard.

È possibile aggiungere file all'archivio di test prima di caricarlo AWS Device Farm utilizzando il seguente comando:

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

Per una directory di file aggiuntivi:

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

Questi comandi funzionano come previsto per tutti i formati di caricamento dei pacchetti di test ad eccezione dei file IPA. Per i file IPA, specialmente se utilizzati con XCUITests, si consiglia di inserire i file aggiuntivi in una posizione leggermente diversa a causa della progettazione dei pacchetti AWS Device Farm di test iOS. Quando crei il tuo test iOS, la directory dell'applicazione di test si troverà all'interno di un'altra directory denominata *Payload*.

Ad esempio, ecco come potrebbe apparire una di queste directory di test iOS:

```
$ tree
.
```

```

### Payload
  ### ADFiOSReferenceAppUITests-Runner.app
    ### ADFiOSReferenceAppUITests-Runner
    ### Frameworks
    #   ### XCTAutomationSupport.framework
    #   #   ### Info.plist
    #   #   ### XCTAutomationSupport
    #   #   ### _CodeSignature
    #   #   #   ### CodeResources
    #   #   ### version.plist
    #   ### XCTest.framework
    #       ### Info.plist
    #       ### XCTest
    #       ### _CodeSignature
    #       #   ### CodeResources
    #       ### en.lproj
    #       #   ### InfoPlist.strings
    #       ### version.plist
    ### Info.plist
    ### PkgInfo
    ### PlugIns
    #   ### ADFiOSReferenceAppUITests.xctest
    #   #   ### ADFiOSReferenceAppUITests
    #   #   ### Info.plist
    #   #   ### _CodeSignature
    #   #   ### CodeResources
    #   ### ADFiOSReferenceAppUITests.xctest.dSYM
    #       ### Contents
    #       ### Info.plist
    #       ### Resources
    #       ### DWARF
    #       ### ADFiOSReferenceAppUITests
    ### _CodeSignature
    #   ### CodeResources
    ### embedded.mobileprovision

```

Per questi XCUITest pacchetti, aggiungete eventuali file aggiuntivi alla directory che termina all'.*app* interno della *Payload* directory. Ad esempio, i comandi seguenti mostrano come aggiungere un file a questo pacchetto di test:

```

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

```

Quando aggiungete un file al pacchetto di test, potete aspettarvi un comportamento di interazione leggermente diverso in AWS Device Farm base al formato di caricamento. Se il caricamento ha utilizzato l'estensione del file ZIP, AWS Device Farm decomprimerà automaticamente il caricamento prima del test e lascerà i file decompressi nella posizione con la `$DEVICEFARM_TEST_PACKAGE_PATH` variabile di ambiente. (Ciò significa che se aggiungete un file chiamato *extra_file* alla radice dell'archivio come nel primo esempio, esso si troverebbe in `$DEVICEFARM_TEST_PACKAGE_PATH/extra_file` durante il test).

Per usare un esempio più pratico, se sei un utente di Appium TestNG che desidera includere un *testng.xml* file nel test, puoi includerlo nel tuo archivio usando il seguente comando:

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

Quindi, puoi modificare il comando test nella modalità ambiente personalizzata nel modo seguente:

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

Se l'estensione per il caricamento del pacchetto di test non è ZIP (ad esempio, APK, IPA o JAR), il file del pacchetto caricato si trova in `$DEVICEFARM_TEST_PACKAGE_PATH`. Poiché si tratta ancora di file in formato di archivio, puoi decomprimere il file per accedere ai file aggiuntivi dall'interno. Ad esempio, il comando seguente decomprimerà il contenuto del pacchetto di test (per file APK, IPA o JAR) nella directory: */tmp*

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

Nel caso di un file APK o JAR, i file aggiuntivi verranno decompressi */tmp* nella directory (ad esempio,). */tmp/extra_file* Nel caso di un file IPA, come spiegato in precedenza, i file aggiuntivi si troverebbero in una posizione leggermente diversa all'interno della cartella che termina con *.app*, che si trova all'interno della directory. *Payload* Ad esempio, in base all'esempio IPA precedente, il file si troverebbe nella posizione */tmp/Payload/ADFiOSReferenceAppUITests-Runner.app/extra_file* (referenziabile come). */tmp/Payload/*.app/extra_file*

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

Accesso remoto in AWS Device Farm

L'accesso remoto consente di eseguire gesti, operazioni di scorrimento e di interagire con un dispositivo tramite browser Web in tempo reale per testare la funzionalità e riprodurre i problemi dei clienti. Puoi interagire con un dispositivo specifico creando una sessione di accesso remoto con quel dispositivo.

Una sessione in Device Farm è un'interazione in tempo reale con un dispositivo fisico effettivo ospitato in un browser web. Una sessione mostra il singolo dispositivo selezionato all'avvio della sessione. Un utente può avviare più di una sessione alla volta con il numero totale di dispositivi simultanei tenendo conto del limite del numero di slot dei dispositivi di cui dispone. Puoi acquistare slot dei dispositivi in base alla famiglia di dispositivi (Android o iOS). Per ulteriori informazioni, consulta [Prezzi di Device Farm](#).

Device Farm offre attualmente un sottoinsieme di dispositivi per il test di accesso remoto. Nuovi dispositivi vengono costantemente aggiunti al pool di dispositivi.

Device Farm acquisisce il video di ogni sessione di accesso remoto e genera registri delle attività durante la sessione. I risultati includono qualsiasi informazione che fornisci durante una sessione.

Note

Per motivi di sicurezza, ti consigliamo di non fornire o inserire informazioni sensibili come numeri di conti, login personale e altri dati durante una sessione di accesso remoto. Se possibile, utilizza alternative sviluppate specificamente per i test, come gli account di test.

Argomenti

- [Creazione di una sessione di accesso remoto in AWS Device Farm](#)
- [Utilizzo di una sessione di accesso remoto in AWS Device Farm](#)
- [Recupero dei risultati di una sessione di accesso remoto in AWS Device Farm](#)

Creazione di 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 \(console\)](#)
- [Passaggi successivi](#)

Prerequisiti

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

Creare una sessione con la console Device Farm

1. Accedere alla console Device Farm all'indirizzo <https://console.aws.amazon.com/devicefarm>.
2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.
3. Se hai già un progetto, selezionalo dall'elenco. Altrimenti, crea un progetto seguendo le istruzioni riportate in [Creazione di 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).

Passaggi successivi

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

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.

Utilizzo di una sessione di accesso remoto in AWS Device Farm

Per informazioni sull'esecuzione dei test interattivi delle app Android e iOS tramite le sessioni di accesso remoto, consulta [Sessioni](#).

- [Prerequisiti](#)
- [Utilizzare una sessione nella console Device Farm](#)
- [Passaggi successivi](#)
- [Suggerimenti e trucchi](#)

Prerequisiti

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

Utilizzare una sessione nella console Device Farm

Quando il dispositivo che hai richiesto per una sessione di accesso remoto diventa disponibile, la console mostra lo schermo del dispositivo. La sessione ha una durata massima di 150 minuti. Il tempo rimanente nella sessione viene visualizzato nel campo Tempo rimasto accanto al nome del dispositivo.

Installazione di un'applicazione

Per installare un'applicazione sul dispositivo di sessione, in Installa applicazioni, seleziona Scegli file, quindi scegli il file.apk (Android) o il file.ipa (iOS) che desideri installare. Le applicazioni che esegui in una sessione di accesso remoto non richiedono alcuna strumentazione di test o provisioning.

 Note

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.

Passaggi successivi

Device Farm continua la sessione finché non viene interrotta manualmente o non viene raggiunto il limite di 150 minuti. Per terminare la sessione, scegli Arresta sessione. Una volta interrotta la sessione puoi accedere al video acquisito e ai log generati. Per ulteriori informazioni, consulta [Recupero dei risultati delle sessioni dalle sessioni di accesso remoto](#).

Suggerimenti e trucchi

In alcune AWS regioni potrebbero verificarsi problemi di prestazioni con la sessione di accesso remoto. Ciò è dovuto in parte alla latenza che si verifica in alcune regioni. In caso di problemi a livello di prestazioni, concedi alla sessione di accesso remoto il tempo di recuperare prima di interagire nuovamente con l'app.

Recupero dei risultati di una sessione di accesso remoto in AWS Device Farm

Per informazioni sulle sessioni, consulta [Sessioni](#).

- [Prerequisiti](#)
- [Visualizzazione dei dettagli della sessione](#)
- [Scaricamento del video o dei registri della sessione](#)

Prerequisiti

- Completa una sessione. Segui le istruzioni riportate in [Utilizzo di una sessione di accesso remoto in AWS Device Farm](#), poi torna a questa pagina.

Visualizzazione dei dettagli della sessione

Al termine di una sessione di accesso remoto, la console Device Farm visualizza una tabella che contiene dettagli sull'attività durante la sessione. Per ulteriori informazioni, consulta [Analisi delle informazioni di log](#).

Per tornare ai dettagli di una sessione in un secondo momento:

1. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.
2. Scegliete il progetto contenente la sessione.
3. Scegli Accesso remoto, quindi scegli la sessione che desideri rivedere dall'elenco.

Scaricamento del video o dei registri della sessione

Al termine di una sessione di accesso remoto, la console Device Farm fornisce l'accesso a un'acquisizione video della sessione e dei registri delle attività. Nei risultati della sessione, selezionare la scheda Files (File) per un elenco di collegamenti ai log e video della sessione. È possibile visualizzare questi file all'interno del browser o salvarli in locale.

Dispositivi privati in AWS Device Farm

Un dispositivo privato è un dispositivo mobile fisico che AWS Device Farm distribuisce per tuo conto in un data center Amazon. Questo dispositivo è esclusivo per il tuo AWS account.

Note

Attualmente, i dispositivi privati sono disponibili solo nella regione AWS Stati Uniti occidentali (Oregon) (us-west-2).

Se possiedi un parco istanze di dispositivi privati, è possibile creare sessioni di accesso remoto e pianificare sessioni di test tramite i dispositivi privati. Per ulteriori informazioni, consulta [Creazione di un'esecuzione di test o avvio di una sessione di accesso remoto in AWS Device Farm](#). È anche possibile creare profili di istanza per controllare il comportamento dei dispositivi privati durante una sessione di accesso remoto o di test. Per ulteriori informazioni, consulta [Creazione di un profilo di istanza in AWS Device Farm](#). Facoltativamente, puoi richiedere che determinati dispositivi privati Android vengano distribuiti come dispositivi rootati.

Puoi anche creare un servizio endpoint Amazon Virtual Private Cloud per testare app private a cui la tua azienda ha accesso ma che non sono raggiungibili tramite Internet. Ad esempio, potresti avere un'applicazione Web in esecuzione sulla tua VPC che desideri testare su dispositivi mobili. Per ulteriori informazioni, consulta [Utilizzo dei servizi endpoint Amazon VPC con Device Farm - Legacy \(non consigliato\)](#).

Se sei interessato a utilizzare una flotta di dispositivi privati, [contattaci](#). Il team di Device Farm deve collaborare con te per configurare e distribuire una flotta di dispositivi privati per il tuo AWS account.

Argomenti

- [Creazione di un profilo di istanza in AWS Device Farm](#)
- [Richiedi dispositivi privati aggiuntivi in AWS Device Farm](#)
- [Creazione di un'esecuzione di test o avvio di una sessione di accesso remoto in AWS Device Farm](#)
- [Selezione di dispositivi privati in un pool di dispositivi in AWS Device Farm](#)
- [Ignorare la nuova firma delle app su dispositivi privati in AWS Device Farm](#)
- [Amazon VPC in tutte le AWS regioni in AWS Device Farm](#)
- [Disattivazione di dispositivi privati in Device Farm](#)

Creazione di un profilo di istanza in AWS Device Farm

È possibile impostare un parco istanze che contiene uno o più dispositivi privati. Questi dispositivi sono dedicati al tuo account AWS . Dopo aver configurato i dispositivi, è possibile creare uno o più profili di istanza per loro. I profili di istanza consentono di automatizzare le sessioni di test e di applicare le stesse impostazioni in maniera coerente alle istanze del dispositivo. I profili delle istanze possono anche aiutarti a controllare il comportamento della sessione di accesso remoto. Per ulteriori informazioni sui dispositivi privati in Device Farm, vedere [Dispositivi privati in AWS Device Farm](#).

Per creare un'istanza

1. Apri la console Device Farm all'indirizzo <https://console.aws.amazon.com/devicefarm/>.
2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Dispositivi privati.
3. Scegliere Instance profiles (Profili delle istanze).
4. Seleziona Crea profilo dell'istanza.
5. Immettere un nome per il profilo dell'istanza.

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. Seleziona Salva.

Richiedi dispositivi privati aggiuntivi in AWS Device Farm

In AWS Device Farm, puoi richiedere l'aggiunta di ulteriori istanze di dispositivi privati alla tua flotta. Puoi anche visualizzare e modificare le impostazioni delle istanze di dispositivi privati esistenti nella tua flotta. Per ulteriori informazioni sui dispositivi privati, consulta [Dispositivi privati in AWS Device Farm](#).

Per richiedere dispositivi privati aggiuntivi o modificarne le impostazioni

1. Apri la console Device Farm all'indirizzo <https://console.aws.amazon.com/devicefarm/>.
2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Dispositivi privati.
3. Scegliere Device instances (Istanze dispositivo). La scheda Device instances (Istanze dispositivo) visualizza una tabella dei dispositivi private del parco istanze. Per cercare o filtrare rapidamente la tabella, inserisci i termini di ricerca nella barra di ricerca sopra le colonne.
4. Per richiedere una nuova istanza di dispositivo privato, scegli Richiedi istanza di dispositivo o [contattaci](#). I dispositivi privati richiedono una configurazione aggiuntiva con l'aiuto del team di Device Farm.
5. Nella tabella delle istanze del dispositivo, scegli l'opzione di attivazione/disattivazione accanto all'istanza su cui desideri visualizzare o gestire le informazioni, quindi scegli Modifica.

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

- Per allegare un profilo di istanza all'istanza del dispositivo, selezionalo dall'elenco a discesa Profilo. Ad esempio, allegare un profilo di istanza può essere utile se desideri escludere sempre un pacchetto di app specifico dalle attività di pulizia. Per ulteriori informazioni sull'utilizzo dei profili di istanza con i dispositivi, consulta. [Creazione di un profilo di istanza in AWS Device Farm](#)
- (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.
- Seleziona Salva.

Creazione di un'esecuzione di test o avvio di una sessione di accesso remoto in AWS Device Farm

In AWS Device Farm, dopo aver configurato una flotta di dispositivi privati, puoi creare esecuzioni di test o avviare sessioni di accesso remoto con uno o più dispositivi privati della tua flotta. Per ulteriori informazioni sui dispositivi privati, consulta [Dispositivi privati in AWS Device Farm](#).

Per creare un test, eseguire o avviare una sessione di accesso remoto

1. Apri la console Device Farm all'indirizzo <https://console.aws.amazon.com/devicefarm/>.
2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.
3. Scegli un progetto esistente dall'elenco o creane uno nuovo. Per creare un nuovo progetto, scegliete Nuovo progetto, inserite un nome per il progetto, quindi scegliete Invia.
4. Esegui una 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)

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

< 1 2 >

	Name	Status	Platform	OS	Form factor	Instance Id	Labels
☐	OnePlus 8T	AVAILABLE	Android	11	Phone	-	-
☐	Samsung Galaxy Tab S7	AVAILABLE	Android	11	Tablet	-	-

Selezione di dispositivi privati in un pool di dispositivi in AWS Device Farm

Per utilizzare dispositivi privati durante il test, puoi creare un pool di dispositivi che seleziona i tuoi dispositivi privati. I pool di dispositivi consentono di selezionare dispositivi privati principalmente attraverso tre tipi di regole del pool di dispositivi:

1. Regole basate sull'ARN del dispositivo
2. Regole basate sull'etichetta dell'istanza del dispositivo
3. Regole basate sull'ARN dell'istanza del dispositivo

Nelle sezioni seguenti, ogni tipo di regola e i relativi casi d'uso sono descritti in modo approfondito. È possibile utilizzare la console Device Farm, l'interfaccia a riga di AWS comando (AWS CLI) o l'API Device Farm per creare o modificare un pool di dispositivi con dispositivi privati utilizzando queste regole.

Argomenti

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

- [Creazione di un pool di dispositivi privati con dispositivi privati \(API\)](#)

ARN del dispositivo

L'ARN di un dispositivo è un identificatore che rappresenta un tipo di dispositivo anziché un'istanza fisica specifica del dispositivo. Un tipo di dispositivo è definito dai seguenti attributi:

- L'ID della flotta del dispositivo
- L'OEM del dispositivo
- Il numero di modello del dispositivo
- La versione del sistema operativo del dispositivo
- Lo stato del dispositivo che indica se è rootato o meno

Molte istanze di dispositivo fisico possono essere rappresentate da un singolo tipo di dispositivo in cui ogni istanza di quel tipo ha gli stessi valori per questi attributi. Ad esempio, se hai tre *Apple iPhone 13* dispositivi con versione iOS *16.1.0* nella tua flotta privata, ogni dispositivo condividerebbe lo stesso ARN del dispositivo. Se dal tuo parco dispositivi venissero aggiunti o rimossi gli stessi attributi, l'ARN del dispositivo continuerebbe a rappresentare tutti i dispositivi disponibili nel tuo parco dispositivi per quel tipo di dispositivo.

L'ARN dei dispositivi è il modo più efficace per selezionare i dispositivi privati per un pool di dispositivi perché consente al pool di dispositivi di continuare a selezionare i dispositivi indipendentemente dalle istanze specifiche del dispositivo che hai distribuito in un dato momento. Le singole istanze di dispositivi privati possono subire guasti hardware, pertanto Device Farm è tenuta a sostituirle automaticamente con nuove istanze funzionanti dello stesso tipo di dispositivo. In questi scenari, la regola ARN del dispositivo garantisce che il pool di dispositivi possa continuare a selezionare i dispositivi in caso di guasto hardware.

Quando utilizzi una regola ARN del dispositivo per i dispositivi privati nel tuo pool di dispositivi e pianifichi un'esecuzione di test con quel pool, Device Farm verificherà automaticamente quali istanze di dispositivo privato sono rappresentate dall'ARN di quel dispositivo. Tra le istanze attualmente disponibili, una di esse verrà assegnata per eseguire il test. Se al momento non sono disponibili istanze, Device Farm attenderà che la prima istanza disponibile dell'ARN di quel dispositivo diventi disponibile e la assegnerà per eseguire il test.

Etichette delle istanze del dispositivo

L'etichetta di un'istanza di dispositivo è un identificatore testuale che è possibile allegare come metadati per un'istanza di dispositivo. È possibile allegare più etichette a ciascuna istanza del dispositivo e la stessa etichetta a più istanze del dispositivo. Per ulteriori informazioni sull'aggiunta, la modifica o la rimozione delle etichette dei dispositivi dalle istanze del dispositivo, consulta [Gestione dei dispositivi privati](#).

L'etichetta dell'istanza del dispositivo può essere un modo efficace per selezionare i dispositivi privati per un pool di dispositivi perché, se si dispone di più istanze di dispositivo con la stessa etichetta, consente al pool di dispositivi di selezionare una di esse per il test. Se l'ARN del dispositivo non è una buona regola per il tuo caso d'uso (ad esempio, se desideri scegliere tra dispositivi di più tipi di dispositivi o se desideri selezionare da un sottoinsieme di tutti i dispositivi di un tipo di dispositivo), le etichette delle istanze del dispositivo possono consentirti di scegliere tra più dispositivi per il tuo pool di dispositivi con maggiore granularità. Le singole istanze di dispositivi privati possono subire guasti hardware, pertanto Device Farm è tenuta a sostituirle automaticamente con nuove istanze funzionanti dello stesso tipo di dispositivo. In questi scenari, l'istanza del dispositivo sostitutivo non conserverà alcun metadato dell'etichetta di istanza del dispositivo sostituito. Pertanto, se applichi la stessa etichetta di istanza di dispositivo a più istanze di dispositivo, la regola di etichettatura delle istanze del dispositivo garantisce che il pool di dispositivi possa continuare a selezionare le istanze del dispositivo in caso di guasto hardware.

Quando utilizzi una regola di etichettatura delle istanze di dispositivo per i dispositivi privati nel tuo pool di dispositivi e pianifichi un'esecuzione di test con quel pool, Device Farm verificherà automaticamente quali istanze di dispositivo privato sono rappresentate da quell'etichetta di istanza del dispositivo e, tra quelle istanze, ne seleziona casualmente una disponibile per eseguire il test. Se non ce ne sono disponibili, Device Farm selezionerà casualmente qualsiasi istanza del dispositivo con l'etichetta dell'istanza del dispositivo per eseguire il test e metterà in coda il test da eseguire sul dispositivo una volta che sarà disponibile.

ARN istanza

L'ARN di un'istanza di dispositivo è un identificatore che rappresenta un'istanza fisica di dispositivo bare metal distribuita in una flotta privata. Ad esempio, se aveste tre *iPhone 13* dispositivi con sistema operativo *15.0.0* nella vostra flotta privata, mentre ogni dispositivo condividesse lo stesso ARN del dispositivo, ogni dispositivo avrebbe anche il proprio ARN di istanza che rappresenta solo quell'istanza.

L'ARN dell'istanza del dispositivo è il modo meno affidabile per selezionare i dispositivi privati per un pool di dispositivi ed è consigliato solo se le etichette del dispositivo ARNs e dell'istanza del dispositivo non si adattano al tuo caso d'uso. Le istanze di dispositivo vengono spesso utilizzate come regole per i pool di dispositivi quando un'istanza di dispositivo specifica è configurata in modo unico e specifico come prerequisito per il test e se è necessario conoscere e verificare tale configurazione prima di eseguire il test su di essa. Le singole istanze di dispositivi privati possono subire guasti hardware, pertanto Device Farm è tenuta a sostituirle automaticamente con nuove istanze funzionanti dello stesso tipo di dispositivo. In questi scenari, l'istanza del dispositivo sostitutiva avrà un ARN dell'istanza del dispositivo diverso rispetto al dispositivo sostituito. Quindi, se ti affidi all'istanza del dispositivo ARNs per il tuo pool di dispositivi, dovrai modificare manualmente la definizione delle regole del pool di dispositivi dall'utilizzo del vecchio ARN all'utilizzo del nuovo ARN. Se è necessario preconfigurare manualmente il dispositivo per il test, questo può essere un flusso di lavoro efficace (rispetto al dispositivo). Per eseguire test su larga scala, si consiglia di provare ad adattare questi casi d'uso alle etichette delle istanze del dispositivo e, se possibile, di preconfigurare più istanze del dispositivo per il test.

Quando utilizzi una regola ARN di istanza di dispositivo per i dispositivi privati nel tuo pool di dispositivi e pianifichi un'esecuzione di test con quel pool, Device Farm assegnerà automaticamente quel test a quell'istanza del dispositivo. Se l'istanza del dispositivo non è disponibile, Device Farm metterà in coda il test sul dispositivo non appena sarà disponibile.

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

Quando si crea una sessione di test, è possibile creare un pool di dispositivi per la sessione di test e per assicurare che il pool includa solo i propri dispositivi privati.

Note

Quando si crea un pool di dispositivi con dispositivi privati nella console, è possibile utilizzare solo una delle tre regole disponibili per la selezione dei dispositivi privati. Se desideri creare un pool di dispositivi che contenga più tipi di regole per dispositivi privati (ad esempio, pool di dispositivi che contengono regole per il dispositivo ARNs e l'istanza del dispositivo ARNs), devi creare il pool tramite la CLI o l'API.

1. Apri la console Device Farm all'indirizzo <https://console.aws.amazon.com/devicefarm/>.
2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.

3. Scegli un progetto esistente dall'elenco o creane uno nuovo. Per creare un nuovo progetto, scegliete Nuovo progetto, inserite un nome per il progetto, quindi scegliete Invia.
4. Scegliete Impostazioni del progetto, quindi passate alla scheda Device pool.
5. Scegli Crea pool di dispositivi e inserisci un nome e una descrizione opzionale per il tuo pool di dispositivi.
 - a. Per utilizzare le regole ARN dei dispositivi per il tuo pool di dispositivi, scegli Crea pool di dispositivi statico, quindi seleziona i tipi di dispositivi specifici dall'elenco che desideri utilizzare nel pool di dispositivi. Non selezionate solo istanze di dispositivo private perché questa opzione fa sì che il pool di dispositivi venga creato con le regole ARN delle istanze del dispositivo (anziché le regole ARN del dispositivo).

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
Ⓐ	Available	Android	10	Phone		-

Cancel Create

- 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 comeField, scegli Contiene come e specifica l'Operatoreetichetta dell'istanza del dispositivo desiderata comeValue.

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

Filter by device attribute
 Use filters to create a dynamic device pool. We recommend creating device pools with an "Availability" filter so your tests don't wait for devices that are being used by other customers.

Field	Operator	Value
Instance Labels	CONTAINS	Example

Max devices
 Enter max number of devices

Mobile devices (0/92)

Name	Status	Platform	OS	Form factor	Instance Id	Labels

- c. Per utilizzare le regole ARN delle istanze di dispositivo per il tuo pool di dispositivi, scegli Crea pool di dispositivi statico, quindi seleziona Solo istanze di dispositivo private per limitare l'elenco dei dispositivi alle sole istanze di dispositivi privati che Device Farm ha associato al tuo account. AWS

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)

Name	Status	Platform	OS	Form factor	Instance Id	Labels
🛡️	Available	Android	10	Phone		-

6. Scegli Create (Crea).

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

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

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

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

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

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

Ignorare la nuova firma delle app su dispositivi privati in AWS Device Farm

La firma delle app è un processo che prevede la firma digitale di un pacchetto di app (ad esempio [APK](#), [IPA](#)) con una chiave privata prima che possa essere installato su un dispositivo o pubblicato su un app store come Google Play Store o Apple App Store. Per semplificare i test riducendo il numero di firme e profili necessari e aumentare la sicurezza dei dati sui dispositivi remoti, AWS Device Farm firmerà nuovamente l'app dopo averla caricata sul servizio.

Una volta caricata l'app su AWS Device Farm, il servizio genererà una nuova firma per l'app utilizzando i propri certificati di firma e profili di provisioning. Questo processo sostituisce la firma originale dell'app con la firma di AWS Device Farm. L'app rifirmata viene quindi installata sui dispositivi di test forniti da AWS Device Farm. La nuova firma consente all'app di essere installata ed eseguita su questi dispositivi senza la necessità dei certificati originali dello sviluppatore.

Su iOS, sostituiamo il profilo di provisioning 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 della VPN.

Su Android, abbandoniamo l'app. Ciò potrebbe interrompere funzionalità che dipendono dalla firma dell'app, come l'API Android di Google Maps. Può inoltre attivare il rilevamento antipirateria e antimanomissione disponibile in prodotti come DexGuard. Per i test integrati, possiamo modificare il manifesto per includere le autorizzazioni necessarie per acquisire e salvare schermate.

Quando usi dispositivi privati, puoi saltare la fase in cui AWS Device Farm rifirma la tua app. Questo è diverso dai dispositivi pubblici, in cui Device Farm riassegna sempre la firma dell'app sulle piattaforme Android e iOS.

È possibile ignorare la nuova firma dell'app al momento della creazione di una sessione di accesso remoto o di test. Questo può essere utile se la funzionalità dell'app si interrompe quando Device Farm firma nuovamente l'app. Ad esempio, le notifiche push potrebbero non funzionare dopo la nuova firma. Per ulteriori informazioni sulle modifiche apportate da Device Farm durante il test dell'app, consulta la pagina [AWS Device Farm FAQs](#) o [Apps](#).

Per saltare la nuova firma dell'app per un'esecuzione di test, seleziona Ignora la nuova firma dell'app in Configurazione aggiuntiva. Questa opzione è disponibile solo per i dispositivi privati.

▼ **Additional configuration**

Video recording
If checked, enables video recording during test execution.
☒ Enable video recording

App performance
If checked, enables capture of performance data from the device.
☐ Enable app performance data capture

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

Add extra data
Upload extra data
Upload a .zip file to be extracted before your app is tested.

or drop file here

Note

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

Le fasi aggiuntive per configurare le impostazioni della firma delle app variano in base all'utilizzo di dispositivi privati Android o iOS.

Ignorare la nuova firma dell'app sui dispositivi Android

Se stai testando l'app su un dispositivo privato Android, seleziona Skip app re-signing (Ignora nuova firma dell'app) quando crei una sessione di test o di accesso remoto. Non è richiesta alcuna configurazione aggiuntiva.

Ignorare la nuova firma dell'app sui dispositivi iOS

Apple richiede di firmare un'app per l'esecuzione di test prima di caricarla su un dispositivo. Per i dispositivi iOS sono disponibili due opzioni per la firma dell'app.

- Se si sta utilizzando un profilo per sviluppatori interno (Enterprise), è possibile passare alla sezione successiva [the section called “Crea una sessione di accesso remoto per rendere affidabile la tua app”](#).
- Se utilizzi un profilo di sviluppo di app iOS ad hoc, è necessario prima registrare il dispositivo con il tuo account sviluppatore Apple e aggiornare il profilo di provisioning in modo che includa il dispositivo privato. È necessario quindi firmare nuovamente la tua app con il profilo di provisioning che hai aggiornato. È quindi possibile eseguire l'app rifirmata in Device Farm.

Come eseguire la registrazione di un dispositivo con un profilo di provisioning di sviluppo di app iOS ad hoc

1. Eseguire l'accesso al proprio account sviluppatori Apple.
2. Vai alla sezione Certificati e profili della console. IDs
3. Vai a Devices (Dispositivi).
4. Registrare il dispositivo nel proprio account sviluppatore Apple. Per ottenere il nome e l'UDID del dispositivo, utilizza il `ListDeviceInstances` funzionamento dell'API Device Farm.
5. Andare al proprio profilo di provisioning e scegliere Edit (Modifica).
6. Scegliere il dispositivo dall'elenco.
7. In XCode, recuperare il proprio profilo di provisioning aggiornato e firmare nuovamente l'app.

Non è richiesta alcuna configurazione aggiuntiva. Ora puoi creare una sessione di accesso remoto o di test e selezionare Skip app re-signing (Ignora nuova firma dell'app).

Creazione di una sessione di accesso remoto per fidarsi della tua app iOS

Se utilizzi un profilo di provisioning per sviluppatori in-house (enterprise), devi eseguire una sola volta la procedura di conferma dell'attendibilità del certificato dello sviluppatore dell'app in-house su ciascuno dei tuoi dispositivi privati.

A 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 di istanza in AWS Device Farm](#) per creare o modificare un profilo di istanza in Device Farm. Nel profilo dell'istanza, aggiungi l'ID del pacchetto dell'app 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 [Creazione di una sessione](#) per creare una sessione di accesso remoto utilizzando il profilo dell'istanza del dispositivo privato creato. Quando si crea la sessione, assicurarsi di selezionare Skip app re-signing (Ignora rifirma dell'app).

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.

Amazon VPC in tutte le AWS regioni in AWS Device Farm

I servizi Device Farm sono disponibili solo nella regione Stati Uniti occidentali (Oregon) (us-west-2). Puoi usare Amazon Virtual Private Cloud (Amazon VPC) per raggiungere un servizio nel tuo Amazon Virtual Private Cloud in un'altra AWS regione utilizzando Device Farm. Se Device Farm e il tuo servizio si trovano nella stessa regione, vedi [Utilizzo dei servizi endpoint Amazon VPC con Device Farm - Legacy \(non consigliato\)](#).

Esistono due modi per accedere ai servizi privati situati in una regione diversa. Se disponi di servizi situati in un'altra regione diversa us-west-2, puoi utilizzare VPC Peering per peerizzare il VPC di quella regione su un altro VPC che si interfaccia con Device Farm in us-west-2. Tuttavia, se disponi di servizi in più regioni, un Transit Gateway ti consentirà di accedere a tali servizi con una configurazione di rete più semplice.

Per ulteriori informazioni, consulta gli [scenari di peering VPC](#) nella Amazon VPC Peering Guide.

Panoramica del peering VPC per diverse regioni VPCs in AWS Device Farm

Puoi peerizzarne due VPCs in regioni diverse purché abbiano blocchi CIDR distinti e non sovrapposti. Ciò garantisce che tutti gli indirizzi IP privati siano unici e consente a tutte le risorse presenti di VPCs indirizzarsi tra loro senza la necessità di alcuna forma di traduzione degli indirizzi di rete (NAT). Per ulteriori informazioni sulla notazione CIDR, consulta [RFC 4632](#).

Questo argomento include uno scenario di esempio interregionale in cui Device Farm (denominata VPC-1) si trova nella regione Stati Uniti occidentali (Oregon) (us-west-2). Il secondo VPC in questo esempio (denominato VPC-2) si trova in un'altra regione.

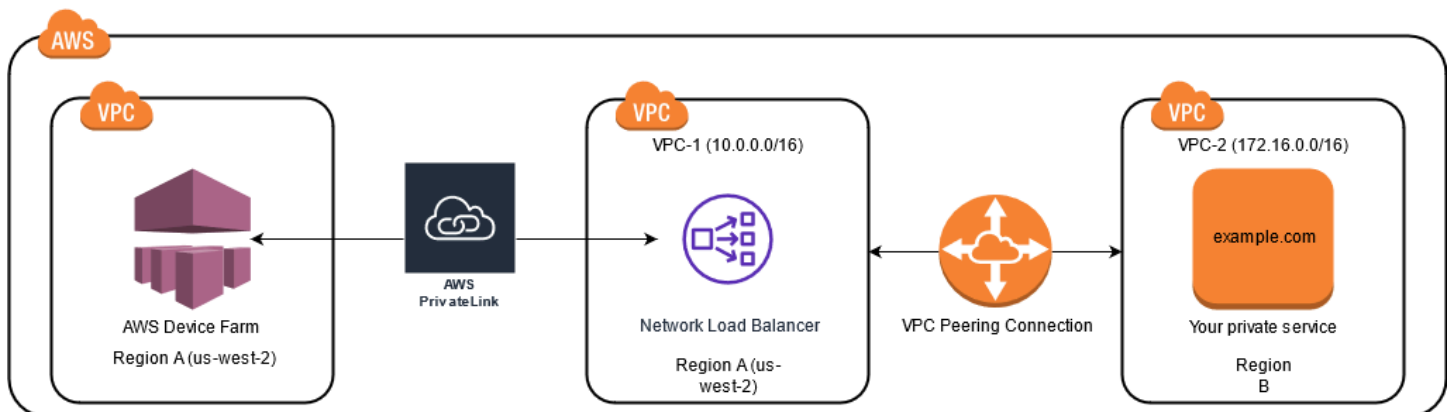
Esempio di Device Farm VPC interregionale

Componente VPC	VPC-1	VPC-2
CIDR	10.0.0.0/16	172.16.0.0/16

⚠ Important

La creazione di una connessione peering tra due VPCs può modificare il livello di sicurezza di VPCs. Inoltre, l'aggiunta di nuove voci alle relative tabelle di routing può modificare il livello di sicurezza delle risorse all'interno di VPCs. È responsabilità dell'utente implementare queste configurazioni in modo da soddisfare i requisiti di sicurezza dell'organizzazione. Per ulteriori informazioni, consulta il [modello di responsabilità condivisa](#).

Il seguente diagramma mostra i componenti nell'esempio e le interazioni tra i componenti.

**Argomenti**

- [Prerequisiti per l'utilizzo di Amazon VPC in AWS Device Farm](#)
- [Fase 1: Configurazione di una connessione peering tra VPC-1 e VPC-2](#)
- [Fase 2: Aggiornamento delle tabelle di routing in VPC-1 e VPC-2](#)
- [Fase 3: Creazione di un gruppo target](#)
- [Fase 4: Creazione di un Network Load Balancer](#)
- [Fase 5: Creazione di un servizio endpoint VPC per connettere il VPC a Device Farm](#)
- [Fase 6: Crea una configurazione di endpoint VPC tra il tuo VPC e Device Farm](#)
- [Passaggio 7: creazione di un'esecuzione di test per utilizzare la configurazione dell'endpoint VPC](#)
- [Creazione di una rete scalabile con Transit Gateway](#)

Prerequisiti per l'utilizzo di Amazon VPC in AWS Device Farm

Questo esempio richiede quanto segue:

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

Per ulteriori informazioni sulla creazione VPCs e la configurazione di sottoreti, consulta [Working with VPCs and subnet](#) nella Amazon VPC Peering Guide.

Fase 1: Configurazione di una connessione peering tra VPC-1 e VPC-2

Stabilire una connessione peering tra i due blocchi CIDR VPCs contenenti blocchi CIDR non sovrapposti. A tale scopo, consulta [Creare e accettare connessioni peering VPC nella Amazon VPC Peering](#) Guide. Utilizzando lo scenario interregionale di questo argomento e la Amazon VPC Peering Guide, viene creato il seguente esempio di configurazione della connessione peering:

Nome

Device-Farm-Peering-Connection-1

ID VPC (richiedente)

vpc-0987654321gfedcba (VPC-2)

Account

My account

Region

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

ID VPC (Accetta)

vpc-1234567890abcdefg (VPC-1)

Note

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

Fase 2: Aggiornamento delle tabelle di routing in VPC-1 e VPC-2

Dopo aver impostato una connessione peering, è necessario stabilire un percorso di destinazione tra i due VPCs per trasferire i dati tra di loro. Per stabilire questa route, è possibile aggiornare manualmente la tabella delle rotte di VPC-1 in modo che punti alla sottorete di VPC-2 e viceversa. A tale scopo, consulta [Aggiornare le tabelle di routing per una connessione peering VPC nella Amazon VPC Peering Guide](#). Utilizzando lo scenario interregionale di questo argomento e la Amazon VPC Peering Guide, viene creato il seguente esempio di configurazione della tabella di routing:

Esempio di tabella di routing VPC di Device Farm

Componente VPC	VPC-1	VPC-2
ID della tabella di routing	rtb-1234567890abcdefg	rtb-0987654321gfedcba
Intervallo di indirizzi locali	10.0.0.0/16	172.16.0.0/16
Intervallo di indirizzi di destinazione	172.16.0.0/16	10.0.0.0/16

Fase 3: Creazione di un gruppo target

Dopo aver impostato i percorsi di destinazione, è possibile configurare un Network Load Balancer in VPC-1 per indirizzare le richieste a VPC-2.

Il Network Load Balancer deve innanzitutto contenere un gruppo target contenente gli indirizzi IP a cui vengono inviate le richieste.

Per creare un gruppo target

1. Identifica gli indirizzi IP del servizio che desideri utilizzare come target in VPC-2.
 - Questi indirizzi IP devono essere membri della sottorete utilizzata nella connessione peering.
 - Gli indirizzi IP di destinazione devono essere statici e immutabili. Se il tuo servizio dispone di indirizzi IP dinamici, valuta la possibilità di indirizzare le richieste a una risorsa statica (ad esempio un Network Load Balancer) e 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 Amazon <https://console.aws.amazon.com/ec2/> all'indirizzo, quindi scegli Istanze.
- Se ti rivolgi a 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 i servizi in un VPC diverso richiedono la seguente configurazione:

- Per Scegli un tipo di destinazione, scegli gli indirizzi IP.
- Per VPC, scegli il VPC che ospiterà il load balancer. Per l'esempio dell'argomento, questo sarà VPC-1.
- Nella pagina Registra destinazioni, registra una destinazione per ogni indirizzo IP in VPC-2.

Per Rete, scegli Altro indirizzo IP privato.

Per Zona di disponibilità, scegli le zone desiderate in VPC-1.

Per IPv4 l'indirizzo, scegli l'indirizzo IP VPC-2.

Per Porte, scegli le tue porte.

- Seleziona Includi come in 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

IPv4 address

172.16.100.60

Porte

80

Fase 4: Creazione di un Network Load Balancer

Creare un Network Load Balancer utilizzando il gruppo target descritto nel [passaggio 3](#). A tale scopo, vedere [Creazione di un Network Load Balancer](#).

Utilizzando lo scenario interregionale di questo argomento, i seguenti valori vengono utilizzati in una configurazione di esempio di Network Load Balancer:

Load balancer name (Nome del sistema di bilanciamento del carico)

my-nlb

Schema

Internal

VPC

vpc-1234567890abcdefg (VPC-1)

Mappatura

us-west-2a - subnet-4i23iuufkdiuufsloi

us-west-2b - subnet-7x989pkjj78nmn23j

us-west-2c - subnet-0231ndmas12bnnsds

Protocollo/porta

TCP : 80

Gruppo bersaglio

my-target-group

Fase 5: Creazione di un servizio endpoint VPC per connettere il VPC a Device Farm

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

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

Fase 6: Crea una configurazione di endpoint VPC tra il tuo VPC e Device Farm

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

Utilizzando lo scenario interregionale di questo argomento, i seguenti valori vengono utilizzati in una configurazione di endpoint VPC di esempio:

Nome

My VPCE Configuration

Nome del servizio VPCE

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

Nome DNS del servizio

```
devicefarm.com
```

Passaggio 7: creazione di un'esecuzione di test per utilizzare la configurazione dell'endpoint VPC

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

Creazione di una rete scalabile con Transit Gateway

Per creare una rete scalabile utilizzandone più di due VPCs, puoi utilizzare Transit Gateway per fungere da hub di transito di rete per interconnettere le tue reti VPCs e quelle locali. Per configurare un VPC nella stessa regione di Device Farm per utilizzare un Transit Gateway, puoi seguire la guida Amazon [VPC endpoint services with Device Farm](#) per indirizzare le risorse in un'altra regione in base ai loro indirizzi IP privati.

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

Disattivazione di dispositivi privati in Device Farm

<Per disattivare un dispositivo privato dopo il periodo iniziale concordato, è n
Per ulteriori informazioni sui dispositivi privati, consulta. [Dispositivi privati in AWS Device Farm](#)

Important

Queste istruzioni si applicano solo alla risoluzione dei contratti relativi ai dispositivi privati. Per tutti gli altri AWS servizi e problemi di fatturazione, consulta la documentazione relativa a tali prodotti o contatta AWS l'assistenza.

VPC-ENI nella Device Farm di AWS

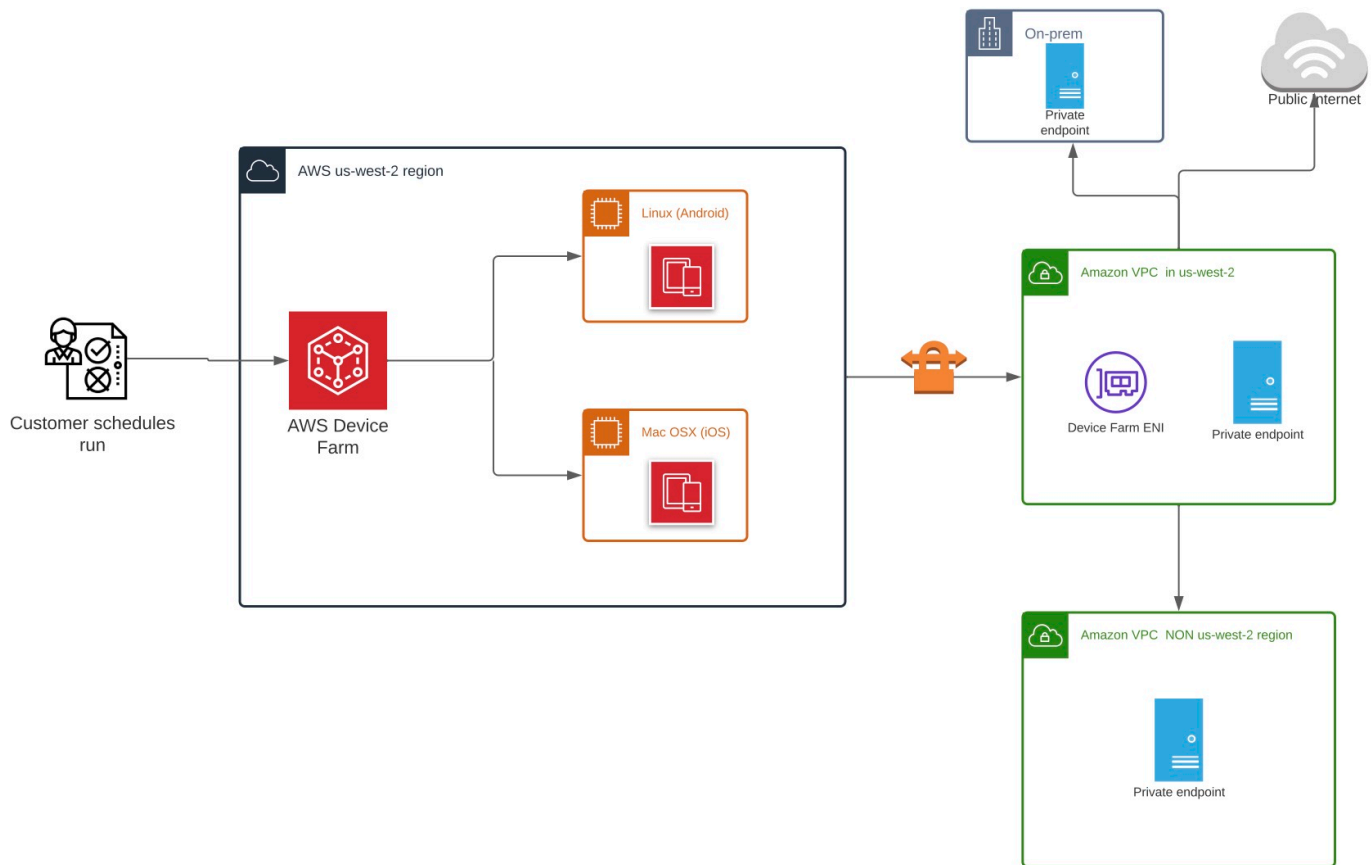
Warning

Questa funzionalità è disponibile solo su dispositivi privati. Per richiedere l'uso privato del dispositivo sul tuo AWS account, [contattaci](#). Se hai già aggiunto dispositivi privati al tuo AWS account, ti consigliamo vivamente di utilizzare questo metodo di connettività VPC.

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

Puoi connettere sia i dispositivi mobili Device Farm che le relative macchine host a un ambiente Amazon Virtual Private Cloud (Amazon VPC) nella us-west-2 regione, che consente l'accesso a non-internet-facing servizi e applicazioni isolati tramite un'interfaccia di [rete elastica](#). Per ulteriori informazioni VPCs, consulta la [Amazon VPC User Guide](#).

Se il tuo endpoint o VPC privato non si trova us-west-2 nella regione, puoi collegarlo a un VPC us-west-2 della regione utilizzando soluzioni come Transit Gateway o VPC Peering. In tali situazioni, Device Farm creerà un ENI in una sottorete fornita per il VPC us-west-2 della regione e sarete responsabili di garantire che sia possibile stabilire una connessione tra il VPC us-west-2 della regione e il VPC dell'altra regione.



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

Note

Device Farm non addebita alcun costo per la creazione ENIs nel VPC di un cliente. us-west-2 Il costo per la connettività inter-VPC esterna o interregionale non è incluso in questa funzionalità.

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

non sia presente un gateway NAT specificato all'interno del VPC. Per ulteriori informazioni, consulta [Gateway NAT](#) nella Guida per l'utente di Amazon VPC.

Argomenti

- [AWS controllo degli accessi e IAM](#)
- [Ruoli collegati ai servizi](#)
- [Prerequisiti](#)
- [Connessione ad Amazon VPC](#)
- [Limiti](#)
- [Utilizzo dei servizi endpoint Amazon VPC 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 policy che garantiscono o limitano l'accesso alle funzionalità di Device Farm. Per utilizzare la funzionalità di connettività VPC con AWS Device Farm, è richiesta la seguente policy IAM per l'account utente o il ruolo che stai utilizzando per accedere ad 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 configurazione VPC, la policy IAM deve consentire di eseguire le seguenti azioni sulle risorse elencate nella configurazione VPC:

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

Inoltre, la tua policy IAM deve consentire anche la creazione del ruolo collegato al servizio:

```
"iam:CreateServiceLinkedRole"
```

Note

Nessuna di queste autorizzazioni è richiesta per gli utenti che non utilizzano configurazioni VPC nei propri progetti.

Ruoli collegati ai servizi

AWS Device Farm utilizza AWS Identity and Access Management ruoli [collegati ai servizi](#) (IAM). Un ruolo collegato ai servizi è un tipo unico di ruolo IAM collegato direttamente a Device Farm. I ruoli collegati ai servizi sono predefiniti da Device Farm e includono tutte le autorizzazioni richieste dal servizio per chiamare altri AWS servizi per conto dell'utente.

Un ruolo collegato al servizio semplifica la configurazione di Device Farm perché non è necessario aggiungere manualmente le autorizzazioni necessarie. Device Farm definisce le autorizzazioni dei suoi ruoli collegati ai servizi e, se non diversamente definito, solo Device Farm può assumerne i ruoli. Le autorizzazioni definite includono la policy di attendibilità e la policy delle autorizzazioni che non può essere allegata a nessun'altra entità IAM.

È possibile eliminare un ruolo collegato 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 sugli altri servizi che supportano i ruoli collegati ai servizi, consulta [Servizi AWS supportati da IAM](#) e cerca i servizi che riportano Sì nella colonna Ruolo associato ai servizi. Scegli un Sì con un link per visualizzare la documentazione relativa al ruolo collegato ai servizi per tale servizio.

Autorizzazioni di ruolo collegate ai servizi per Device Farm

Device Farm utilizza il ruolo collegato al servizio denominato `AWSServiceRoleForDeviceFarm`: consente a Device Farm di accedere alle risorse AWS per tuo conto.

Il ruolo `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
 - Modificare 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 policy IAM 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"
    }
  }
}
]
}
```

Per consentire a un'entità IAM (come un utente, un gruppo o un ruolo) di creare, modificare o eliminare un ruolo collegato ai servizi devi configurare le relative autorizzazioni. Per ulteriori informazioni, consulta [Autorizzazioni del ruolo collegato ai servizi](#) nella Guida per l'utente di IAM.

Creazione di un ruolo collegato al servizio per Device Farm

Quando fornisci una configurazione VPC per un progetto di test mobile, non è necessario creare manualmente un ruolo collegato al servizio. Quando crei la tua prima risorsa Device Farm nella AWS Management Console AWS CLI, o nell' AWS API, Device Farm crea automaticamente il ruolo collegato al servizio.

Se elimini questo ruolo collegato ai servizi, è possibile ricrearlo seguendo lo stesso processo utilizzato per ricreare il ruolo nell'account. Quando crei la tua prima risorsa Device Farm, Device Farm crea nuovamente il ruolo collegato al servizio per te.

Puoi anche utilizzare la console IAM per creare un ruolo collegato al servizio con lo use case Device Farm. Nell'API AWS CLI o nell' AWS API, crea un ruolo collegato al servizio con il nome del servizio. `devicefarm.amazonaws.com` Per ulteriori informazioni, consulta [Creazione di un ruolo collegato ai servizi](#) nella Guida per l'utente IAM. Se elimini il ruolo collegato ai servizi, è possibile utilizzare lo stesso processo per crearlo nuovamente.

Modifica di un ruolo collegato al servizio per Device Farm

Device Farm non consente di modificare il ruolo `AWSService RoleForDeviceFarm` collegato al servizio. Dopo aver creato un ruolo collegato al servizio, non potrai modificarne il nome perché varie entità potrebbero farvi riferimento. È possibile tuttavia modificarne la descrizione utilizzando IAM. Per ulteriori informazioni, consulta la sezione [Modifica di un ruolo collegato ai servizi](#) nella Guida per l'utente di IAM.

Eliminazione di un ruolo collegato al servizio per Device Farm

Se non è più necessario utilizzare una funzionalità o un servizio che richiede un ruolo collegato al servizio, ti consigliamo di eliminare il ruolo. In questo modo non sarà più presente un'entità non utilizzata che non viene monitorata e gestita attivamente. Tuttavia, è necessario effettuare la pulizia delle risorse associate al ruolo collegato al servizio prima di poterlo eliminare manualmente.

Note

Se il servizio Device Farm utilizza il ruolo quando si tenta di eliminare le risorse, l'eliminazione potrebbe non riuscire. In questo caso, attendi alcuni minuti e quindi ripeti l'operazione.

Per eliminare manualmente il ruolo collegato ai servizi mediante IAM

Utilizza la console IAM AWS CLI, il o l' AWS API per eliminare il ruolo AWSService RoleForDeviceFarm collegato al servizio. Per ulteriori informazioni, consulta [Eliminazione del ruolo collegato ai servizi](#) nella Guida per l'utente di 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](#).

Device Farm non supporta l'utilizzo di ruoli collegati al servizio in tutte le aree geografiche in cui il servizio è disponibile. È possibile utilizzare il AWSService RoleForDeviceFarm ruolo nelle seguenti regioni.

Nome della Regione	Identità della regione	Support in Device Farm
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 della 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 configurazioni VPC-ENI:

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

È necessario verificare di avere accesso al VPC specificato quando si configura la connessione. È necessario configurare determinate autorizzazioni Amazon Elastic Compute Cloud (Amazon EC2) per Device Farm.

- La risoluzione DNS è richiesta nel VPC che utilizzi.

- Una volta creato il VPC, avrai bisogno delle seguenti informazioni sul VPC della regione: us-west-2
 - ID VPC
 - Subnet IDs (solo sottoreti private)
 - Gruppo di sicurezza IDs
- È necessario configurare le connessioni Amazon VPC in base al progetto. Al momento, puoi configurare solo una configurazione VPC per progetto. Quando configuri un VPC, Amazon VPC crea un'interfaccia all'interno del tuo VPC e la assegna alle sottoreti e ai gruppi di sicurezza specificati. Tutte le sessioni future associate al progetto utilizzeranno la connessione VPC configurata.
- Non è possibile utilizzare le configurazioni VPC-ENI insieme alla funzionalità VPCE precedente.
- Consigliamo vivamente di non aggiornare un progetto esistente con una configurazione VPC-ENI poiché i progetti esistenti potrebbero avere impostazioni VPCE che persistono a livello di esecuzione. Invece, se utilizzi già le funzionalità VPCE esistenti, usa VPC-ENI per tutti i nuovi progetti.

Connessione ad Amazon VPC

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

- L'ID VPC è inserito us-west-2 se l'app è ospitata lì o l'ID us-west-2 VPC che si connette a un altro VPC in un'altra regione.
- I gruppi di sicurezza applicabili da applicare alla connessione.
- Le sottoreti che verranno associate alla connessione. All'avvio di una sessione, viene utilizzata la sottorete più grande disponibile. Ti consigliamo di associare più sottoreti a diverse zone di disponibilità per migliorare la posizione di disponibilità della connettività VPC.

Dopo aver creato la configurazione VPC-ENI, puoi aggiornarne i dettagli utilizzando la console o la CLI seguendo i passaggi seguenti.

Console

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

2. Nel pannello di navigazione di Device Farm, scegli Mobile Device Testing, quindi scegli Progetti.
3. In Mobile Testing projects, scegli il nome del tuo progetto dall'elenco.
4. Selezionare Project settings (Impostazioni del progetto).
5. Nella sezione Impostazioni Virtual Private Cloud (VPC), puoi modificare VPC, Subnets (solo sottoreti private) e Security Groups.
6. Seleziona Salva.

CLI

Utilizza il seguente comando AWS CLI per aggiornare Amazon VPC:

```
$ aws devicefarm update-project \
--arn arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef \
--vpc-config \
securityGroupIds=sg-02c1537701a7e3763,sg-005dadf9311efda25,\
subnetIds=subnet-09b1a45f9cac53717,subnet-09b1a45f9cac12345,\
vpcId=vpc-0238fb322af81a368
```

Puoi anche configurare un Amazon VPC durante la creazione del tuo progetto:

```
$ aws devicefarm create-project \
--name VPCDemo \
--vpc-config \
securityGroupIds=sg-02c1537701a7e3763,sg-005dadf9311efda25,\
subnetIds=subnet-09b1a45f9cac53717,subnet-09b1a45f9cac12345,\
vpcId=vpc-0238fb322af81a368
```

Limiti

Le seguenti limitazioni sono applicabili alla funzionalità VPC-ENI:

- Puoi fornire fino a cinque gruppi di sicurezza nella configurazione VPC di un progetto Device Farm.
- È possibile fornire fino a otto sottoreti nella configurazione VPC di un progetto Device Farm.
- Quando configuri un progetto Device Farm per utilizzarlo con il tuo VPC, la sottorete più piccola che puoi fornire deve avere almeno cinque indirizzi disponibili. IPv4

- Al momento gli indirizzi IP pubblici non sono supportati. Ti consigliamo invece di utilizzare sottoreti private nei tuoi progetti Device Farm. Se hai bisogno di un accesso pubblico a Internet durante i test, utilizza un gateway [NAT \(Network Address Translation\)](#). La configurazione di un progetto Device Farm con una sottorete pubblica non fornisce ai test l'accesso a Internet o un indirizzo IP pubblico.
- L'integrazione VPC-ENI supporta solo sottoreti private nel tuo VPC.
- È supportato solo il traffico in uscita dall'ENI gestito dal servizio. Ciò significa che l'ENI non può ricevere richieste in entrata non richieste dal VPC.

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

Warning

Consigliamo vivamente di utilizzare la connettività VPC-ENI descritta in [questa](#) pagina per la connettività endpoint privata poiché VPCE è ora considerata una funzionalità legacy. VPC-ENI offre maggiore flessibilità, configurazioni più semplici, è più efficiente in termini di costi e richiede un sovraccarico di manutenzione significativamente inferiore rispetto al metodo di connettività VPCE.

Note

L'utilizzo di Amazon VPC Endpoint Services con Device Farm è supportato solo per i clienti con dispositivi privati configurati. Per consentire al tuo account AWS di utilizzare questa funzionalità con dispositivi privati, [contattaci](#).

Amazon Virtual Private Cloud (Amazon VPC) è un AWS servizio che puoi utilizzare per avviare AWS risorse in una rete virtuale definita dall'utente. Con un VPC, hai il controllo sulle impostazioni di rete, come l'intervallo di indirizzi IP, le sottoreti, le tabelle di routing e i gateway di rete.

Se utilizzi Amazon VPC per ospitare applicazioni private nella AWS regione Stati Uniti occidentali (Oregon) (us-west-2), puoi stabilire una connessione privata tra il tuo VPC e Device Farm. Con questa connessione, è possibile utilizzare Device Farm per testare applicazioni private senza

esporle attraverso la rete Internet pubblica. Per consentire al tuo AWS account di utilizzare questa funzionalità con dispositivi privati, [contattaci](#).

Per connettere una risorsa nel tuo VPC a Device Farm, puoi utilizzare la console Amazon VPC per creare un servizio endpoint VPC. Questo servizio endpoint ti consente di fornire la risorsa del tuo VPC a Device Farm tramite un endpoint VPC Device Farm. Il servizio endpoint fornisce una connettività affidabile e scalabile a Device Farm senza richiedere un gateway Internet, un'istanza NAT (Network Address Translation) o una connessione VPN. Per ulteriori informazioni, consulta [VPC endpoint services \(AWS PrivateLink\) nella Guida](#).AWS PrivateLink

Important

La funzionalità endpoint VPC di Device Farm ti aiuta a connettere in modo sicuro i servizi interni privati del tuo VPC al VPC pubblico di Device Farm utilizzando le connessioni. AWS PrivateLink Anche se la connessione è sicura e privata, tale sicurezza dipende dalla protezione delle proprie credenziali AWS . Se AWS le tue credenziali sono compromesse, un utente malintenzionato può accedere ai dati del servizio o esporli al mondo esterno.

Dopo aver creato un servizio di endpoint VPC in Amazon VPC, puoi utilizzare la console Device Farm per creare una configurazione di endpoint VPC in Device Farm. Questo argomento mostra come creare la connessione Amazon VPC e la configurazione degli endpoint VPC in Device Farm.

Prima di iniziare

Le seguenti informazioni sono per gli utenti di Amazon VPC nella regione Stati Uniti occidentali (Oregon) (us-west-2), con una sottorete in ciascuna delle seguenti zone di disponibilità: us-west-2a, us-west-2b e us-west-2c.

Device Farm ha requisiti aggiuntivi per i servizi endpoint VPC con cui puoi utilizzarlo. Quando crei e configuri un servizio endpoint VPC per funzionare con Device Farm, assicurati di scegliere opzioni che soddisfino i seguenti requisiti:

- Le zone di disponibilità per il servizio devono includere us-west-2a, us-west-2b e us-west-2c. Il Network Load Balancer associato a un servizio endpoint VPC determina le zone di disponibilità per quel servizio endpoint VPC. Se il tuo servizio endpoint VPC non mostra tutte e tre queste zone di disponibilità, devi ricreare il Network Load Balancer per abilitare queste tre zone e quindi riassociare Network Load Balancer al servizio endpoint.

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

Inoltre, se mantieni attiva l'impostazione Accettazione richiesta quando crei il servizio endpoint VPC, devi accettare manualmente ogni richiesta di connessione che Device Farm invia al servizio endpoint. Per modificare questa impostazione per un servizio endpoint esistente, scegli il servizio endpoint sulla console Amazon VPC, scegli Azioni, quindi scegli Modifica impostazione di accettazione degli endpoint. Per ulteriori informazioni, consulta [Modifica dei sistemi di bilanciamento del carico e delle impostazioni di accettazione nella Guida](#).AWS PrivateLink

La sezione successiva spiega come creare un servizio endpoint Amazon VPC che soddisfi questi requisiti.

Fase 1: Creazione di un Network Load Balancer

Il primo passo per stabilire una connessione privata tra il VPC e Device Farm consiste nel creare un Network Load Balancer per indirizzare le richieste a un gruppo target.

New console

Per creare un Network Load Balancer utilizzando la nuova console

1. Apri la console Amazon Elastic Compute Cloud (Amazon EC2) all'indirizzo <https://console.aws.amazon.com/ec2/>.
2. Nel pannello di navigazione, in Load balancing, scegli Load balancers.
3. Selezionare Create Load Balancer (Crea sistema di bilanciamento del carico).
4. In Network load balancer, scegli Crea.
5. Nella pagina Crea sistema di bilanciamento del carico di rete, in Configurazione di base, procedi come segue:
 - a. Immettete il nome di un load balancer.
 - b. Per Schema, scegli Interno.
6. In Network mappings (Mappature di rete), esegui le operazioni seguenti:
 - a. Scegli il VPC per il tuo gruppo target.

- b. Seleziona le seguenti mappature:
 - us-west-2a
 - us-west-2b
 - us-west-2c
7. In Listener and routing, utilizzate le opzioni Protocollo e Porta per scegliere il gruppo target.

 Note

Per impostazione predefinita, il bilanciamento del carico tra zone di disponibilità è disabilitato.

Poiché il sistema di bilanciamento del carico utilizza le zone us-west-2a di disponibilità e us-west-2c richiede la registrazione degli obiettivi in ciascuna di tali zone di disponibilità oppure, se si registrano i target in meno di tutte e tre le zone, è necessario abilitare il bilanciamento del carico tra zone. us-west-2b In caso contrario, il sistema di bilanciamento del carico potrebbe non funzionare come previsto.

8. Selezionare Create Load Balancer (Crea sistema di bilanciamento del carico).

Old console

Per creare un Network Load Balancer utilizzando la vecchia console

1. Apri la console Amazon Elastic Compute Cloud (Amazon EC2) all'indirizzo <https://console.aws.amazon.com/ec2/>.
2. Nel pannello di navigazione, in Load balancing, scegli Load balancer.
3. Selezionare Create Load Balancer (Crea sistema di bilanciamento del carico).
4. In Network load balancer, scegli Crea.
5. Nella pagina Configura il bilanciamento del carico, in Configurazione di base, procedi come segue:
 - a. Inserisci il nome del 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 il VPC per il tuo gruppo target.
 - b. Seleziona le seguenti zone di disponibilità:
 - us-west-2a
 - us-west-2b
 - us-west-2c
 - c. Scegli Avanti: configura le impostazioni di sicurezza.
8. (Facoltativo) Configura le impostazioni di sicurezza, quindi scegli Avanti: configura il routing.
 9. Nella pagina Configure Routing (Configurazione dell'instradamento), procedere come segue:
 - a. Per Target group (Gruppo di destinazione), scegliere Existing target group (Gruppo di destinazione esistente).
 - b. Per Nome, scegli il tuo gruppo target.
 - c. Scegli Avanti: registra gli obiettivi.
 10. Nella pagina Registra obiettivi, esamina gli obiettivi, quindi scegli Avanti: revisione.

 Note

Per impostazione predefinita, il bilanciamento del carico tra zone di disponibilità è disabilitato.

Poiché il sistema di bilanciamento del carico utilizza le zone us-west-2a di disponibilità e us-west-2c richiede la registrazione degli obiettivi in ciascuna di tali zone di disponibilità oppure, se si registrano i target in meno di tutte e tre le zone, è necessario abilitare il bilanciamento del carico tra zone. us-west-2b In caso contrario, il sistema di bilanciamento del carico potrebbe non funzionare come previsto.

11. Controlla la configurazione del sistema di bilanciamento del carico, quindi scegli Crea.

Fase 2: creazione di un servizio endpoint Amazon VPC

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

1. Apri la console Amazon VPC all'indirizzo <https://console.aws.amazon.com/vpc/>.

2. In Risorse per regione, scegli Endpoint services.
3. Scegli Create Endpoint Service (Crea servizio endpoint).
4. Esegui una 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 Amazon VPC.
5. Per le zone di disponibilità incluse us-west-2a us-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 deselecta questa casella di controllo, il servizio di endpoint accetta automaticamente ogni richiesta di connessione che riceve.
7. Scegli Create (Crea) .
8. Nel nuovo servizio endpoint, scegli Allow principal.
9. [Contattaci](#) per ottenere l'ARN dell'endpoint VPC Device Farm (servizio ARN) da aggiungere all'elenco degli indirizzi consentiti per il servizio endpoint, quindi aggiungi l'ARN del servizio all'elenco degli indirizzi consentiti per il servizio.
10. Nella scheda Details (Dettagli) per il servizio di endpoint, annotare il nome del servizio (service name (nome servizio)). Questo nome è necessario quando si crea la configurazione dell'endpoint VPC nella fase successiva.

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

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

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

1. Accedere alla console Device Farm all'indirizzo <https://console.aws.amazon.com/devicefarm>.
2. Nel riquadro di navigazione, scegli Test per dispositivi mobili, quindi Dispositivi privati.

3. Scegli le configurazioni VPCE.
4. Scegli Crea configurazione VPCE.
5. In Crea una nuova configurazione VPCE, inserisci un nome per la configurazione dell'endpoint VPC.
6. Per il nome del servizio VPCE, inserisci il nome del servizio endpoint Amazon VPC (nome del servizio) che hai annotato nella console Amazon VPC. Il nome è simile a `com.amazonaws.vpce.us-west-2.vpce-svc-id`.
7. Per il nome DNS del servizio, inserisci il nome DNS del servizio per l'app che desideri testare (ad esempio, `devicefarm.com`). Non specificare `http` o `https` prima del nome DSN del servizio.

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

8. Seleziona 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'endpoint VPC, puoi utilizzare la configurazione per creare esecuzioni di test o sessioni di accesso remoto. Per ulteriori informazioni, consulta [Creazione di un'esecuzione di test in Device Farm](#) o [Creazione di una sessione](#).

Registrazione delle chiamate API AWS Device Farm con AWS CloudTrail

AWS Device Farm è integrato con AWS CloudTrail un servizio che fornisce un registro delle azioni intraprese da un utente, un ruolo o un AWS servizio in AWS Device Farm. CloudTrail acquisisce tutte le chiamate API per AWS Device Farm come eventi. Le chiamate acquisite includono chiamate dalla console AWS Device Farm e chiamate in codice alle operazioni dell'API AWS Device Farm. Se crei un trail, puoi abilitare la distribuzione continua di CloudTrail eventi a un bucket Amazon S3, inclusi gli eventi per AWS Device Farm. Se non configuri un percorso, puoi comunque visualizzare gli eventi più recenti nella CloudTrail console nella cronologia degli eventi. Utilizzando le informazioni raccolte da CloudTrail, è possibile determinare la richiesta effettuata ad AWS Device Farm, l'indirizzo IP da cui è stata effettuata, chi ha effettuato la richiesta, quando è stata effettuata e ulteriori dettagli.

Per ulteriori informazioni CloudTrail, consulta la [Guida AWS CloudTrail per l'utente](#).

Informazioni su AWS Device Farm in CloudTrail

CloudTrail è abilitato sul tuo AWS account al momento della creazione dell'account. Quando si verifica un'attività in AWS Device Farm, tale attività viene registrata in un CloudTrail evento insieme ad altri eventi di AWS servizio nella cronologia degli eventi. Puoi visualizzare, cercare e scaricare eventi recenti nel tuo AWS account. Per ulteriori informazioni, consulta [Visualizzazione degli eventi con la cronologia degli CloudTrail eventi](#).

Per una registrazione continua degli eventi nel tuo AWS account, inclusi gli eventi per AWS Device Farm, crea un percorso. Un trail consente di CloudTrail inviare file di log a un bucket Amazon S3. Per impostazione predefinita, quando si crea un trail nella console, il trail sarà valido in tutte le regioni AWS. Il trail registra gli eventi di tutte le regioni della AWS partizione e consegna i file di log al bucket Amazon S3 specificato. Inoltre, puoi configurare altri AWS servizi per analizzare ulteriormente e agire in base ai dati sugli eventi raccolti nei log. CloudTrail Per ulteriori informazioni, consulta gli argomenti seguenti:

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

Quando CloudTrail la registrazione è abilitata nell' AWS account, le chiamate API effettuate alle azioni di Device Farm vengono registrate nei file di registro. I record di Device Farm vengono scritti insieme ad altri record di AWS servizio in un file di registro. CloudTrail determina quando creare e scrivere su un nuovo file in base a un periodo di tempo e alle dimensioni del file.

Tutte le azioni di Device Farm vengono registrate e documentate nel [AWS CLI riferimento](#) e nel. [Automazione di Device Farm](#) Ad esempio, le chiamate per creare un nuovo progetto o per l'esecuzione in Device Farm generano voci nei file di CloudTrail registro.

Ogni evento o voce di log contiene informazioni sull'utente che ha generato la richiesta. Le informazioni di identità consentono di determinare quanto segue:

- Se la richiesta è stata effettuata con credenziali utente root o AWS Identity and Access Management (IAM).
- Se la richiesta è stata effettuata con le credenziali di sicurezza temporanee per un ruolo o un utente federato.
- Se la richiesta è stata effettuata da un altro AWS servizio.

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

Informazioni sulle voci dei file di log di AWS Device Farm

Un trail è una configurazione che consente la distribuzione di eventi come file di log in un bucket Amazon S3 specificato dall'utente. CloudTrail i file di registro contengono una o più voci di registro. Un evento rappresenta una singola richiesta proveniente da qualsiasi fonte e include informazioni sull'azione richiesta, la data e l'ora dell'azione, i parametri della richiesta e così via. CloudTrail i file di registro non sono una traccia ordinata dello stack delle chiamate API pubbliche, quindi non vengono visualizzati in un ordine specifico.

L'esempio seguente mostra una voce di CloudTrail registro che dimostra l'ListRunsazione Device Farm:

```
{
  "Records": [
    {
      "eventVersion": "1.03",
      "userIdentity": {
        "type": "Root",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
```

```

    "arn": "arn:aws:iam::123456789012:root",
    "accountId": "123456789012",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2015-07-08T21:13:35Z"
      }
    }
  },
  "eventTime": "2015-07-09T00:51:22Z",
  "eventSource": "devicefarm.amazonaws.com",
  "eventName": "ListRuns",
  "awsRegion": "us-west-2",
  "sourceIpAddress": "203.0.113.11",
  "userAgent": "example-user-agent-string",
  "requestParameters": {
    "arn": "arn:aws:devicefarm:us-west-2:123456789012:project:a9129b8c-df6b-4cdd-8009-40a25EXAMPLE"},
    "responseElements": {
      "runs": [
        {
          "created": "Jul 8, 2015 11:26:12 PM",
          "name": "example.apk",
          "completedJobs": 2,
          "arn": "arn:aws:devicefarm:us-west-2:123456789012:run:a9129b8c-df6b-4cdd-8009-40a256aEXAMPLE/1452d105-e354-4e53-99d8-6c993EXAMPLE",
          "counters": {
            "stopped": 0,
            "warned": 0,
            "failed": 0,
            "passed": 4,
            "skipped": 0,
            "total": 4,
            "errored": 0
          },
          "type": "BUILTIN_FUZZ",
          "status": "RUNNING",
          "totalJobs": 3,
          "platform": "ANDROID_APP",
          "result": "PENDING"
        },
        ... additional entries ...
      ]
    }
  }
}

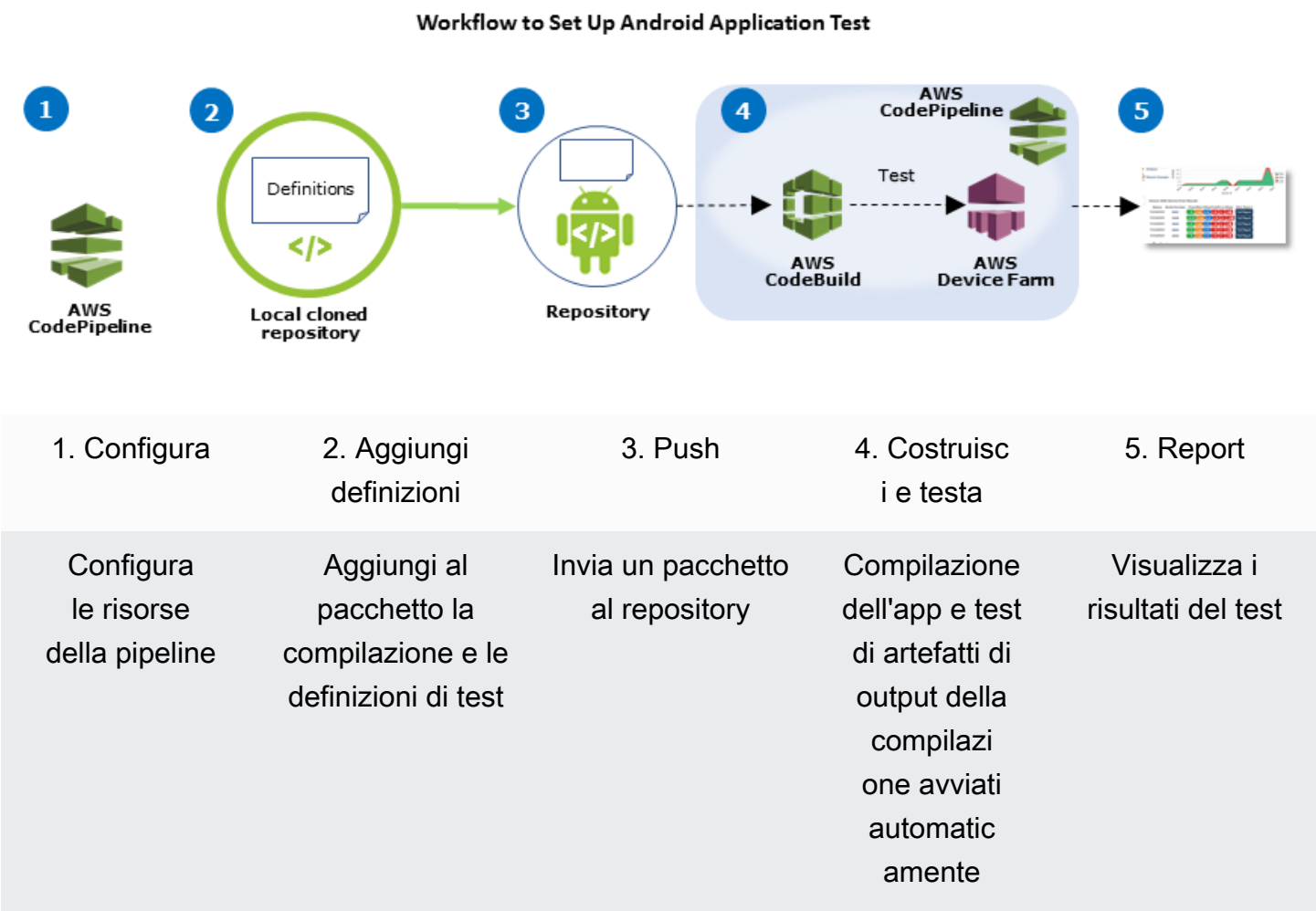
```

```
    }  
  }  
}  
]
```


Integrazione di AWS Device Farm in una fase di CodePipeline test

Puoi utilizzarlo [AWS 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. GitHub



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

Configura CodePipeline per utilizzare i test Device Farm

In questi passaggi, si presuppone che tu abbia [configurato un progetto Device Farm](#) e [creato una pipeline](#). La pipeline deve essere configurata con una fase di test che riceva un [artefatto di input](#) contenente la definizione del tuo test e i file del pacchetto dell'app compilata. L'artefatto di input della fase di test può essere l'artefatto di output di una fase di origine o di compilazione configurata nella tua pipeline.

Per configurare un test di Device Farm, eseguire un CodePipeline test come azione di test

1. Accedi a AWS Management Console e apri la CodePipeline console all'indirizzo <https://console.aws.amazon.com/codepipeline/>.
2. Scegliere la pipeline per la versione della propria app.
3. Nel pannello della fase di test, scegliere l'icona a forma di matita e selezionare quindi Action (Azione).
4. Nel pannello Add action (Aggiungi azione), per Action category (Categoria dell'azione), selezionare Test.
5. Alla voce Action name (Nome operazione), inserire un nome.
6. Alla voce Test provider (Provider del test), scegliere AWS Device Farm.

The screenshot shows the 'Add action' dialog in the AWS CodePipeline console. It includes a close button (X) in the top right corner. Below the title, there is a prompt: 'Choose a serial action from the action category list.' The 'Action category*' dropdown menu is set to 'Test'. Below this, a smaller text says 'Configure how your application is tested.' The 'Test actions' section has a search bar and a list of actions; 'test' is selected. Below the list, the 'Action name*' field is filled with 'test'. At the bottom, the 'Test provider*' dropdown menu is set to 'AWS Device Farm'.

7. In Nome progetto, scegli il tuo progetto Device Farm esistente o scegli Crea un nuovo progetto.

8. Alla voce Device pool (Pool di dispositivi), scegliere il pool di dispositivi preesistenti oppure scegliere Create a new device pool (Crea un nuovo pool di dispositivi). Se crei un pool di dispositivi, devi selezionare un set di dispositivi di test.
9. In App type (Tipo di app), selezionare la piattaforma della propria app.

Device Farm Test

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

Project name*	DemoProject	
	Create a new project	
Device pool*	Top Devices	
	Create a new device pool	
App type*	iOS	
App file path	app-release.apk	
	The location of the application file in your input artifact.	
Test type*	Built-in: Fuzz	
Event count	6000	
	Specify a number between 1 and 10,000, representing the number of user interface events for the fuzz test to perform.	
Event throttle	50	
	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.	
Randomizer seed		
	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.	

10. Alla voce App file path (Percorso file app), inserire il percorso del pacchetto dell'applicazione compilata. Il percorso è relativo alla cartella principale dell'artefatto di input del test.
11. Alla voce Test type (Tipo di test), procedere in uno dei seguenti modi:
 - Se utilizzi uno dei test Device Farm integrati, scegli il tipo di test configurato nel tuo progetto Device Farm.
 - Se non stai utilizzando uno dei test integrati di Device Farm, nel percorso del file di test, inserisci il percorso del file di definizione del test. Il percorso è relativo alla cartella principale dell'artefatto di input del test.

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

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

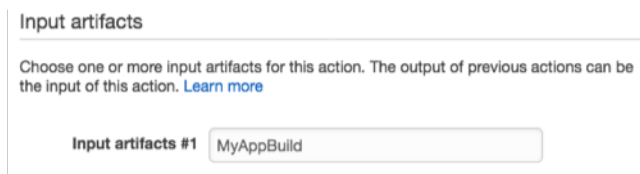
Bluetooth ☒ GPS ☒

NFC ☒ Wifi ☒

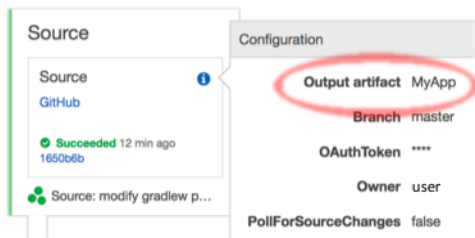
Enable app performance data capture ☒ Enable video recording ☒

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 CodePipeline console, puoi trovare il nome dell'elemento di output per ogni fase passando il mouse sull'icona delle informazioni nel diagramma della pipeline. Se la pipeline verifica l'app direttamente dalla fase Source, scegli. MyApp Se la tua pipeline include una fase di creazione, scegli. MyAppBuild



15. Nella parte inferiore del pannello, scegliere Add Action (Aggiungi operazione).
16. Nel CodePipeline riquadro, scegli Salva modifica alla pipeline, quindi scegli Salva modifica.
17. Per inviare le modifiche e avviare una compilazione tramite pipeline, scegliere Release change (Rilascia modifica) e quindi scegliere Release (Rilascia).

AWS CLI riferimento per AWS Device Farm

Per utilizzare AWS Command Line Interface (AWS CLI) per eseguire i comandi Device Farm, consulta la pagina di [AWS CLI riferimento per AWS Device Farm](#).

Per informazioni generali su AWS CLI, consulta la [Guida per l'AWS Command Line Interface utente](#) e il [AWS CLI Command Reference](#).

PowerShell Riferimento Windows per AWS Device Farm

Per utilizzare Windows per PowerShell eseguire i comandi di Device Farm, vedere il riferimento al [cmdlet Device Farm nella Guida di riferimento](#) ai [AWS Tools for Windows PowerShell cmdlet](#). Per ulteriori informazioni, consulta [Configurazione degli strumenti AWS per Windows PowerShell](#) nella Guida per l'AWS Strumenti per PowerShell utente.

Automazione di AWS Device Farm

L'accesso programmatico a Device Farm è un modo efficace per automatizzare le attività più comuni da svolgere, come la pianificazione di un'esecuzione o il download degli artefatti per un'esecuzione, una suite o un test. L' AWS SDK e provide offrono i mezzi per farlo. AWS CLI

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

- [gli AWS strumenti e SDKs](#)
- [riferimento all'API AWS Device Farm](#)

Esempio: utilizzo dell' AWS SDK per avviare un'esecuzione di Device Farm e raccogliere artefatti

L'esempio seguente fornisce una beginning-to-end dimostrazione di come utilizzare l' AWS SDK per lavorare con Device Farm. Inoltre, vengono effettuate le seguenti operazioni:

- Carica un test e pacchetti applicativi su Device Farm
- Avvia un'esecuzione di test e attende il suo completamento (o errore)
- Scarica tutti gli artefatti prodotti dalle suite di test

Questo esempio dipende dal pacchetto `requests` di terze parti per interagire con HTTP.

```
import boto3
import os
import requests
import string
import random
import time
import datetime
import time
import json

# The following script runs a test through Device Farm
#
# Things you have to change:
config = {
```



```

    # This is our app under test.
    "appFilePath":"app-debug.apk",
    "projectArn": "arn:aws:devicefarm:us-
west-2:111122223333:project:1b99bcff-1111-2222-ab2f-8c3c733c55ed",
    # Since we care about the most popular devices, we'll use a curated pool.
    "testSpecArn":"arn:aws:devicefarm:us-west-2::upload:101e31e8-12ac-11e9-ab14-
d663bd873e83",
    "poolArn":"arn:aws:devicefarm:us-west-2::devicepool:082d10e5-d7d7-48a5-ba5c-
b33d66efa1f5",
    "namePrefix":"MyAppTest",
    # This is our test package. This tutorial won't go into how to make these.
    "testPackage":"tests.zip"
}

client = boto3.client('devicefarm')

unique =
    config['namePrefix']+ "-" + (datetime.date.today().isoformat()) + ('.'.join(random.sample(string.ascii_letters, 10)))

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 AWS Device 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 AWS Device Farm](#)
- [Risoluzione dei problemi delle applicazioni Web Appium Java TestNg in AWS Device Farm](#)
- [Risoluzione dei problemi dei test Appium Python in AWS Device Farm](#)
- [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](#)
- [XCTest Test di risoluzione dei problemi in AWS Device Farm](#)
- [Risoluzione dei problemi dei test XCTest dell'interfaccia utente 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 [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 [Test Android in AWS Device Farm](#).

ANDROID_APP_PACKAGE_NAME_VALUE_MISSING

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

Warning

Impossibile trovare il valore del nome del pacchetto nell'applicazione. Verificare che l'applicazione sia valida eseguendo il comando `aapt debug badging <path to your`

test package> e riprovare dopo la ricerca del valore del nome del pacchetto con parola chiave "pacchetto: nome".

Durante il processo di convalida del caricamento, AWS Device Farm analizza il valore del nome del pacchetto dall'output di un `aapt debug badging <path to your package>` comando.

Controllare che questo comando possa essere eseguito correttamente sull'applicazione Android e che trovi con successo il valore del nome del pacchetto. Nel seguente esempio, il nome del pacchetto è `app-debug.apk`.

- Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

```
$ aapt debug badging app-debug.apk | grep "package: name="
```

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

```
package: name='com.amazon.aws.adf.android.referenceapp' versionCode='1'  
versionName='1.0' platformBuildVersionName='5.1.1-1819727'
```

Per ulteriori informazioni, consulta [Test Android in AWS Device Farm](#).

ANDROID_APP_SDK_VERSION_VALUE_MISSING

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

Warning

Impossibile trovare il valore della versione SDK nell'applicazione. Verificare che l'applicazione sia valida eseguendo il comando `aapt debug badging <path to your test package>` e riprovare dopo la ricerca del valore della versione SDK con parola chiave `sdkVersion`.

Durante il processo di convalida del caricamento, AWS Device Farm analizza il valore della versione dell'SDK dall'output di un comando. `aapt debug badging <path to your package>`

Controllare che questo comando possa essere eseguito correttamente sull'applicazione Android e che trovi con successo il valore del nome del pacchetto. Nel seguente esempio, il nome del pacchetto è app-debug.apk.

- Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

```
$ aapt debug badging app-debug.apk | grep "sdkVersion"
```

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

```
sdkVersion:'9'
```

Per ulteriori informazioni, consulta [Test Android in AWS Device Farm](#).

ANDROID_APP_AAPT_DUMP_XMLTREE_FAILED

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

Warning

Non siamo riusciti a trovare il AndroidManifest file.xml valido nella tua applicazione. Verificare che il pacchetto di test sia valido eseguendo il comando `aapt dump xmltree <path to your test package> AndroidManifest.xml` e riprovare quando il comando non stampa più errori.

Durante il processo di convalida del caricamento, AWS Device Farm analizza le informazioni dall'albero di analisi XML per un file XML contenuto nel pacchetto utilizzando il comando. `aapt dump xmltree <path to your package> AndroidManifest.xml`

Controllare che questo comando possa essere eseguito correttamente sull'applicazione Android. Nel seguente esempio, il nome del pacchetto è app-debug.apk.

- Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

```
$ aapt dump xmltree app-debug.apk. AndroidManifest.xml
```

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

```
N: android=http://schemas.android.com/apk/res/android
E: manifest (line=2)
  A: android:versionCode(0x0101021b)=(type 0x10)0x1
  A: android:versionName(0x0101021c)="1.0" (Raw: "1.0")
  A: package="com.amazon.aws.adf.android.referenceapp" (Raw:
"com.amazon.aws.adf.android.referenceapp")
  A: platformBuildVersionCode=(type 0x10)0x16 (Raw: "22")
  A: platformBuildVersionName="5.1.1-1819727" (Raw: "5.1.1-1819727")
E: uses-sdk (line=7)
  A: android:minSdkVersion(0x0101020c)=(type 0x10)0x9
  A: android:targetSdkVersion(0x01010270)=(type 0x10)0x16
E: uses-permission (line=11)
  A: android:name(0x01010003)="android.permission.INTERNET" (Raw:
"android.permission.INTERNET")
E: uses-permission (line=12)
  A: android:name(0x01010003)="android.permission.CAMERA" (Raw:
"android.permission.CAMERA")
```

Per ulteriori informazioni, consulta [Test Android in AWS Device Farm](#).

ANDROID_APP_DEVICE_ADMIN_PERMISSIONS

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

Warning

L'applicazione richiede autorizzazioni di amministratore del dispositivo. Verificare che le autorizzazioni non siano richieste eseguendo il comando `aapt dump xmltree <path to your test package> AndroidManifest.xml` e riprovare dopo aver controllato che l'output non contenga la parola chiave `android.permission.BIND_DEVICE_ADMIN`.

Durante il processo di convalida del caricamento, AWS Device Farm analizza le informazioni di autorizzazione dall'albero di analisi xml per un file xml contenuto nel pacchetto utilizzando il comando `aapt dump xmltree <path to your package> AndroidManifest.xml`

Verificare che l'applicazione non richieda autorizzazioni di amministratore per il dispositivo. Nel seguente esempio, il nome del pacchetto è app-debug.apk.

- Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

```
$ aapt dump xmltree app-debug.apk AndroidManifest.xml
```

L'output dovrebbe essere come segue:

```
N: android=http://schemas.android.com/apk/res/android
  E: manifest (line=2)
    A: android:versionCode(0x0101021b)=(type 0x10)0x1
    A: android:versionName(0x0101021c)="1.0" (Raw: "1.0")
    A: package="com.amazonaws.devicefarm.android.referenceapp" (Raw:
"com.amazonaws.devicefarm.android.referenceapp")
    A: platformBuildVersionCode=(type 0x10)0x16 (Raw: "22")
    A: platformBuildVersionName="5.1.1-1819727" (Raw: "5.1.1-1819727")
    E: uses-sdk (line=7)
      A: android:minSdkVersion(0x0101020c)=(type 0x10)0xa
      A: android:targetSdkVersion(0x01010270)=(type 0x10)0x16
    E: uses-permission (line=11)
      A: android:name(0x01010003)="android.permission.INTERNET" (Raw:
"android.permission.INTERNET")
    E: uses-permission (line=12)
      A: android:name(0x01010003)="android.permission.CAMERA" (Raw:
"android.permission.CAMERA")
      .....
```

Se l'applicazione Android è valida, l'output non deve contenere: A:
android:name(0x01010003)="android.permission.BIND_DEVICE_ADMIN" (Raw:
"android.permission.BIND_DEVICE_ADMIN").

Per ulteriori informazioni, consulta [Test Android in AWS Device Farm](#).

Alcune finestre della mia applicazione Android mostrano una schermata vuota o nera

Se state testando un'applicazione Android e notate che alcune finestre dell'applicazione appaiono con una schermata nera nella registrazione video del test effettuata da Device Farm, è possibile che l'applicazione utilizzi la FLAG_SECURE funzionalità di Android. Questo flag (come descritto nella [documentazione ufficiale di Android](#)) viene utilizzato per impedire che determinate finestre di un'applicazione vengano registrate dagli strumenti di registrazione dello schermo. Di conseguenza, la funzione di registrazione dello schermo di Device Farm (sia per l'automazione che per i test di accesso remoto) potrebbe mostrare una schermata nera al posto della finestra dell'applicazione se la finestra utilizza questo flag.

Questo flag viene spesso utilizzato dagli sviluppatori per le pagine delle loro applicazioni che contengono informazioni sensibili come le pagine di accesso. Se vedi una schermata nera al posto della schermata dell'applicazione in alcune pagine come la pagina di accesso, collabora con i tuoi sviluppatori per ottenere una versione dell'applicazione che non utilizzi questo flag per i test.

Inoltre, tenete presente che Device Farm può ancora interagire con le finestre delle applicazioni che hanno questo flag. Pertanto, se la pagina di accesso dell'applicazione appare come una schermata nera, è comunque possibile inserire le credenziali per accedere all'applicazione (e quindi visualizzare le pagine non bloccate dal FLAG_SECURE flag).

Risoluzione dei problemi dei JUnit test Java di Appium in AWS Device Farm

L'argomento seguente elenca i messaggi di errore che si verificano durante il caricamento dei JUnit test Java di Appium e consiglia soluzioni alternative per risolvere ogni errore.

Note

Le seguenti istruzioni si basano su Linux x86_64 e Mac.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED

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

 Warning

Impossibile aprire il file ZIP del test. Verificare che il file sia valido e riprovare.

Verificare che sia possibile decomprimere il pacchetto di test senza errori. Nell'esempio seguente, il nome del pacchetto è `.zip`. `zip-with-dependencies`

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip zip-with-dependencies.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Un JUnit pacchetto Appium Java valido dovrebbe produrre un output simile al seguente:

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `- log4j-1.2.14.jar
```

Per ulteriori informazioni, consulta [Test 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 `dependency-jars` sia nel pacchetto e riprovare.

Nell'esempio seguente, il nome del pacchetto è `zip-with-dependencies.zip`.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip zip-with-dependencies.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Se il JUnit pacchetto Appium Java è valido, troverai la *dependency-jars* directory all'interno della directory di lavoro:

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `-- log4j-1.2.14.jar
```

Per ulteriori informazioni, consulta [Test Appium e AWS Device Farm](#).

APPIUM_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DIR

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

 Warning

Impossibile trovare un file JAR nella struttura ad albero delle directory dependency-jars. Decomprimere il pacchetto di test, quindi aprire la directory dependency-jars, verificare che almeno un file JAR si trovi nella directory e riprovare.

Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies .zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip zip-with-dependencies.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Se il JUnit pacchetto Appium Java è valido, troverai almeno un *jar* file all'interno della directory: *dependency-jars*

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `- log4j-1.2.14.jar
```

Per ulteriori informazioni, consulta [Test 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, troverai almeno un *jar* file come *acme-android-appium-1.0-SNAPSHOT-tests.jar* nel nostro esempio. Il nome del file può essere diverso, ma dovrebbe terminare con *-tests.jar*

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `- log4j-1.2.14.jar
```

Per ulteriori informazioni, consulta [Test Appium e AWS Device Farm](#).

APPIUM_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JAR

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

⚠ Warning

Impossibile trovare un file di classe all'interno del file JAR dei test. Decomprimere il pacchetto di test, quindi decomprimere il file JAR dei test e verificare che almeno un file di classe si trovi nel file JAR, quindi riprovare.

Nell'esempio seguente, il nome del pacchetto è `zip-with-dependencies.zip`.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip zip-with-dependencies.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare almeno un file jar come *acme-android-appium-1.0-SNAPSHOT-tests.jar* nel nostro esempio. Il nome del file può essere diverso, ma dovrebbe terminare con *-tests.jar*.

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `-- log4j-1.2.14.jar
```

3. Dopo aver estratto correttamente i file, si dovrebbe trovare almeno una classe nella struttura delle directory di lavoro, eseguendo il comando:

```
$ tree .
```

L'output visualizzato dovrebbe essere di questo tipo:

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- one-class-file.class
|- folder
|   `--another-class-file.class
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `-- log4j-1.2.14.jar
```

Per ulteriori informazioni, consulta [Test 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 file JUnit JAR sia all'interno della directory e riprova.

Nell'esempio seguente, il nome del pacchetto è .zip. zip-with-dependencies

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip zip-with-dependencies.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
tree .
```

L'output dovrebbe essere simile al seguente:

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- junit-4.10.jar
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `-- log4j-1.2.14.jar
```

Se il JUnit pacchetto Appium Java è valido, nel nostro esempio troverai il file delle JUnit dipendenze simile al file *junit-4.10.jar* jar. Il nome deve essere composto dalla parola chiave *junit* e dal relativo numero di versione, che in questo esempio è 4.10.

Per ulteriori informazioni, consulta [Test 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 relativo numero di versione, che nel nostro esempio è 4.10:

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- junit-4.10.jar
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `-- log4j-1.2.14.jar
```

Note

I test potrebbero non essere eseguiti correttamente se la JUnit versione specificata nel pacchetto di test è inferiore alla versione minima 4.10 supportata.

Per ulteriori informazioni, consulta [Test 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

Impossibile aprire il file ZIP del test. Verificare che il file sia valido e riprovare.

Verificare che sia possibile decomprimere il pacchetto di test senza errori. Nell'esempio seguente, il nome del pacchetto è `zip-with-dependencies.zip`.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip zip-with-dependencies.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Un JUnit pacchetto Appium Java valido dovrebbe produrre un output simile al seguente:

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
```

```
|– joda-time-2.7.jar
`– log4j-1.2.14.jar
```

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_DEPENDENCY_DIR_MISSING

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

Warning

Impossibile trovare la directory `dependency-jars` nel pacchetto di test. Decomprimere il pacchetto di test, verificare che la directory `dependency-jars` sia nel pacchetto e riprovare.

Nell'esempio seguente, il nome del pacchetto è `zip-with-dependencies.zip`.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip zip-with-dependencies.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Se il JUnit pacchetto Appium Java è valido, troverai la *dependency-jars* directory all'interno della directory di lavoro:

```
.
|– acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|– acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|– zip-with-dependencies.zip (this .zip file contains all of the items)
`– dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |– com.some-dependency.bar-4.1.jar
    |– com.another-dependency.thing-1.0.jar
    |– joda-time-2.7.jar
    `– log4j-1.2.14.jar
```

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_IN_DEPENDEN

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

Warning

Impossibile trovare un file JAR nella struttura ad albero delle directory dependency-jars. Decomprimere il pacchetto di test, quindi aprire la directory dependency-jars, verificare che almeno un file JAR si trovi nella directory e riprovare.

Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies .zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip zip-with-dependencies.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Se il JUnit pacchetto Appium Java è valido, troverai almeno un *jar* file all'interno della directory: *dependency-jars*

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `- log4j-1.2.14.jar
```

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

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

Warning

Impossibile trovare un file `*-tests.jar` nel pacchetto di test. Decomprimere il pacchetto di test, verificare che nel pacchetto sia presente almeno un file `*-tests.jar` e riprovare.

Nell'esempio seguente, il nome del pacchetto è `zip-with-dependencies.zip`.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip zip-with-dependencies.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Se il JUnit pacchetto Appium Java è valido, troverai almeno un *jar* file come *acme-android-appium-1.0-SNAPSHOT-tests.jar* nel nostro esempio. Il nome del file può essere diverso, ma dovrebbe terminare con *-tests.jar*

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `- log4j-1.2.14.jar
```


APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS

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

Warning

Impossibile trovare un file di classe all'interno del file JAR dei test. Decomprimere il pacchetto di test, quindi decomprimere il file JAR dei test e verificare che almeno un file di classe si trovi nel file JAR, quindi riprovare.

Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip zip-with-dependencies.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare almeno un file jar come *acme-android-appium-1.0-SNAPSHOT-tests.jar* nel nostro esempio. Il nome del file può essere diverso, ma dovrebbe terminare con *-tests.jar*.

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `- log4j-1.2.14.jar
```

3. Dopo aver estratto correttamente i file, si dovrebbe trovare almeno una classe nella struttura delle directory di lavoro, eseguendo il comando:

```
$ tree .
```

L'output visualizzato dovrebbe essere di questo tipo:

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- one-class-file.class
|- folder
|   `-- another-class-file.class
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`-- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `-- log4j-1.2.14.jar
```

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_JUNIT_VERSION_VALUE_UNK

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

Warning

Non siamo riusciti a trovare un valore di JUnit versione. Decomprimi il pacchetto di test e apri la directory `dependency-jars`, verifica che il file JUnit JAR sia all'interno della directory e riprova.

Nell'esempio seguente, il nome del pacchetto è `.zip`. `zip-with-dependencies`

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip zip-with-dependencies.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
tree .
```

L'output dovrebbe essere simile al seguente:

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- junit-4.10.jar
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `-- log4j-1.2.14.jar
```

Se il JUnit pacchetto Appium Java è valido, nel nostro esempio troverai il file delle JUnit dipendenze simile al file *junit-4.10.jar* jar. Il nome deve essere composto dalla parola chiave *junit* e dal relativo numero di versione, che in questo esempio è 4.10.

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION

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

Warning

Abbiamo riscontrato che la JUnit versione era inferiore alla versione minima 4.10 supportata. Cambia la JUnit versione e riprova.

Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip zip-with-dependencies.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare un file di JUnit dipendenza come *junit-4.10.jar* nel nostro esempio e il relativo numero di versione, che nel nostro esempio è 4.10:

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- junit-4.10.jar
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `-- log4j-1.2.14.jar
```

Note

I test potrebbero non essere eseguiti correttamente se la JUnit versione specificata nel pacchetto di test è inferiore alla versione minima 4.10 supportata.

Per ulteriori informazioni, consulta [Test Appium e AWS Device Farm](#).

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

Il seguente argomento elenca i messaggi di errore che si verificano durante il caricamento dei test di Appium Java TestNG e consiglia soluzioni alternative per risolvere ogni errore.

 Note

Le seguenti istruzioni si basano su Linux x86_64 e Mac.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED

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

 Warning

Impossibile aprire il file ZIP del test. Verificare che il file sia valido e riprovare.

Verificare che sia possibile decomprimere il pacchetto di test senza errori. Nell'esempio seguente, il nome del pacchetto è `zip-with-dependencies.zip`.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip zip-with-dependencies.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Un JUnit pacchetto Appium Java valido dovrebbe produrre un output simile al seguente:

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `- log4j-1.2.14.jar
```

Per ulteriori informazioni, consulta [Test 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 la *dependency-jars* directory all'interno della directory di lavoro.

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `-- log4j-1.2.14.jar
```

Per ulteriori informazioni, consulta [Test Appium e AWS Device Farm](#).

APPIUM_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_

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

Warning

Impossibile trovare un file JAR nella struttura ad albero delle directory dependency-jars. Decomprimere il pacchetto di test, quindi aprire la directory dependency-jars, verificare che almeno un file JAR si trovi nella directory e riprovare.

Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies .zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip zip-with-dependencies.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

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

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `-- log4j-1.2.14.jar
```

Per ulteriori informazioni, consulta [Test 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, troverai almeno un *jar* file come *acme-android-appium-1.0-SNAPSHOT-tests.jar* nel nostro esempio. Il nome del file può essere diverso, ma dovrebbe terminare con *-tests.jar*

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `-- log4j-1.2.14.jar
```


Per ulteriori informazioni, consulta [Test Appium e AWS Device Farm](#).

APPIUM_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS

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

Warning

Impossibile trovare un file di classe all'interno del file JAR dei test. Decomprimere il pacchetto di test, quindi decomprimere il file JAR dei test e verificare che almeno un file di classe si trovi nel file JAR, quindi riprovare.

Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip zip-with-dependencies.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare almeno un file jar come *acme-android-appium-1.0-SNAPSHOT-tests.jar* nel nostro esempio. Il nome del file può essere diverso, ma dovrebbe terminare con *-tests.jar*.

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
```

```
`- log4j-1.2.14.jar
```

3. Per estrarre i file dal file jar, eseguire il seguente comando:

```
$ jar xf acme-android-appium-1.0-SNAPSHOT-tests.jar
```

4. Dopo aver estratto correttamente i file, eseguire il comando seguente:

```
$ tree .
```

Dovresti trovare almeno una classe nella struttura ad albero della directory di lavoro:

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- one-class-file.class
|- folder
|   `-- another-class-file.class
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `-- log4j-1.2.14.jar
```

Per ulteriori informazioni, consulta [Test Appium e AWS Device Farm](#).

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

Il seguente argomento elenca i messaggi di errore che si verificano durante il caricamento dei test per applicazioni Web di Appium Java TestNG e consiglia soluzioni alternative per risolvere ogni errore.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED

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

 Warning

Impossibile aprire il file ZIP del test. Verificare che il file sia valido e riprovare.

Verificare che sia possibile decomprimere il pacchetto di test senza errori. Nell'esempio seguente, il nome del pacchetto è `.zip`. `zip-with-dependencies`

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip zip-with-dependencies.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Un JUnit pacchetto Appium Java valido dovrebbe produrre un output simile al seguente:

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `- log4j-1.2.14.jar
```

Per ulteriori informazioni, consulta [Test 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 la *dependency-jars* directory all'interno della directory di lavoro.

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `-- log4j-1.2.14.jar
```

Per ulteriori informazioni, consulta [Test Appium e AWS Device Farm](#).

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DIR

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

⚠ Warning

Impossibile trovare un file JAR nella struttura ad albero delle directory dependency-jars. Decomprimere il pacchetto di test, quindi aprire la directory dependency-jars, verificare che almeno un file JAR si trovi nella directory e riprovare.

Nell'esempio seguente, il nome del pacchetto è zip-with-dependencies .zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip zip-with-dependencies.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

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

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `- log4j-1.2.14.jar
```

Per ulteriori informazioni, consulta [Test 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, troverai almeno un *jar* file come *acme-android-appium-1.0-SNAPSHOT-tests.jar* nel nostro esempio. Il nome del file può essere diverso, ma dovrebbe terminare con *-tests.jar*

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `-- log4j-1.2.14.jar
```

Per ulteriori informazioni, consulta [Test Appium e AWS Device Farm](#).

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JAR

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

⚠ Warning

Impossibile trovare un file di classe all'interno del file JAR dei test. Decomprimere il pacchetto di test, quindi decomprimere il file JAR dei test e verificare che almeno un file di classe si trovi nel file JAR, quindi riprovare.

Nell'esempio seguente, il nome del pacchetto è `zip-with-dependencies.zip`.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip zip-with-dependencies.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare almeno un file jar come *acme-android-appium-1.0-SNAPSHOT-tests.jar* nel nostro esempio. Il nome del file può essere diverso, ma dovrebbe terminare con *-tests.jar*.

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `-- log4j-1.2.14.jar
```

3. Per estrarre i file dal file jar, eseguire il seguente comando:

```
$ jar xf acme-android-appium-1.0-SNAPSHOT-tests.jar
```

4. Dopo aver estratto correttamente i file, eseguire il comando seguente:

```
$ tree .
```

Dovresti trovare almeno una classe nella struttura ad albero della directory di lavoro:

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- one-class-file.class
|- folder
|   `-- another-class-file.class
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`-- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `-- log4j-1.2.14.jar
```

Per ulteriori informazioni, consulta [Test Appium e AWS Device Farm](#).

Risoluzione dei problemi dei test Appium Python in AWS Device Farm

Il seguente argomento elenca i messaggi di errore che si verificano durante il caricamento dei test di Appium Python e consiglia soluzioni alternative per risolvere ogni errore.

APPIUM_PYTHON_TEST_PACKAGE_UNZIP_FAILED

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

Warning

Impossibile aprire il file ZIP del test Appium. Verificare che il file sia valido e riprovare.

Verificare che sia possibile decomprimere il pacchetto di test senza errori. In questo esempio, il nome del pacchetto è `test_bundle.zip`.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip test_bundle.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Un pacchetto Appium Python valido dovrebbe produrre un output come il seguente:

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|   |-- test_unittest.py
|-- wheelhouse (directory)
|   |-- Appium_Python_Client-0.20-cp27-none-any.whl
|   |-- py-1.4.31-py2.py3-none-any.whl
|   |-- pytest-2.9.0-py2.py3-none-any.whl
|   |-- selenium-2.52.0-cp27-none-any.whl
|   |-- wheel-0.26.0-py2.py3-none-any.whl
```

Per ulteriori informazioni, consulta [Test 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, troverai almeno un file `.whl` dipendente come i file evidenziati all'interno della directory. `wheelhouse`

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|   |-- test_unittest.py
|-- wheelhouse (directory)
|   |-- Appium_Python_Client-0.20-cp27-none-any.whl
|   |-- py-1.4.31-py2.py3-none-any.whl
|   |-- pytest-2.9.0-py2.py3-none-any.whl
|   |-- selenium-2.52.0-cp27-none-any.whl
|   |-- wheel-0.26.0-py2.py3-none-any.whl
```

Per ulteriori informazioni, consulta [Test 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, troverai almeno un file `.whl` dipendente come i file evidenziati all'interno della directory. `wheelhouse` Il nome del file può essere diverso, ma dovrebbe terminare con `-any.whl` o `-linux_x86_64.whl`, che specifica la piattaforma. Qualsiasi altra piattaforma, come windows non è supportata.

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|   |-- test_unittest.py
|-- wheelhouse (directory)
|   |-- Appium_Python_Client-0.20-cp27-none-any.whl
|   |-- py-1.4.31-py2.py3-none-any.whl
|   |-- pytest-2.9.0-py2.py3-none-any.whl
|   |-- selenium-2.52.0-cp27-none-any.whl
|   |-- wheel-0.26.0-py2.py3-none-any.whl
```

Per ulteriori informazioni, consulta [Test 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 la `tests` directory all'interno della directory di lavoro.

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|   |-- test_unittest.py
|-- wheelhouse (directory)
|   |-- Appium_Python_Client-0.20-cp27-none-any.whl
|   |-- py-1.4.31-py2.py3-none-any.whl
|   |-- pytest-2.9.0-py2.py3-none-any.whl
|   |-- selenium-2.52.0-cp27-none-any.whl
|   |-- wheel-0.26.0-py2.py3-none-any.whl
```

Per ulteriori informazioni, consulta [Test 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 la `tests` directory all'interno della directory di lavoro. Il nome del file può essere diverso, ma dovrebbe iniziare con `test_` o terminare con `_test.py`.

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|   |-- test_unittest.py
|-- wheelhouse (directory)
|   |-- Appium_Python_Client-0.20-cp27-none-any.whl
|   |-- py-1.4.31-py2.py3-none-any.whl
|   |-- pytest-2.9.0-py2.py3-none-any.whl
|   |-- selenium-2.52.0-cp27-none-any.whl
|   |-- wheel-0.26.0-py2.py3-none-any.whl
```

Per ulteriori informazioni, consulta [Test 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 il *requirements.txt* file all'interno della directory di lavoro.

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|   |-- test_unittest.py
|-- wheelhouse (directory)
|   |-- Appium_Python_Client-0.20-cp27-none-any.whl
|   |-- py-1.4.31-py2.py3-none-any.whl
|   |-- pytest-2.9.0-py2.py3-none-any.whl
|   |-- selenium-2.52.0-cp27-none-any.whl
|   |-- wheel-0.26.0-py2.py3-none-any.whl
```

Per ulteriori informazioni, consulta [Test 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 [Test 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 [Test 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/zhenan/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 [Test 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. Controlla la lunghezza del `requirements.txt` file e il numero di file `.whl` dipendenti nella directory wheelhouse:

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

Se il numero di file `.whl` dipendenti è inferiore al numero di righe non vuote del `requirements.txt` file, devi assicurarti di quanto segue:

- Esiste un file `.whl` dipendente corrispondente a ciascuna riga del `requirements.txt` file.
- Non ci sono altre righe nel `requirements.txt` file che contengono informazioni diverse dai nomi dei pacchetti di dipendenza.
- Nessun nome di dipendenza viene duplicato su più righe del `requirements.txt` file, in modo che due righe del file possano corrispondere a un `.whl` file dipendente.

AWS Device Farm non supporta righe nel `requirements.txt` file che non corrispondono direttamente ai pacchetti di dipendenza, come le righe che specificano le opzioni globali per il `pip install` comando. Consulta [il formato del file dei requisiti](#) per un elenco di opzioni globali.

Per ulteriori informazioni, consulta [Test 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 [Test 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, troverai almeno un file *.whl* dipendente come i file evidenziati all'interno della directory. *wheelhouse*

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|   |-- test_unittest.py
|-- wheelhouse (directory)
|   |-- Appium_Python_Client-0.20-cp27-none-any.whl
|   |-- py-1.4.31-py2.py3-none-any.whl
|   |-- pytest-2.9.0-py2.py3-none-any.whl
|   |-- selenium-2.52.0-cp27-none-any.whl
|   |-- wheel-0.26.0-py2.py3-none-any.whl
```

Per ulteriori informazioni, consulta [Test 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, troverai almeno un file `.whl` dipendente come i file evidenziati all'interno della directory. `wheelhouse` Il nome del file può essere diverso, ma dovrebbe terminare con `-any.whl` o `-linux_x86_64.whl`, che specifica la piattaforma. Qualsiasi altra piattaforma, come windows non è supportata.

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|   |-- test_unittest.py
|-- wheelhouse (directory)
|   |-- Appium_Python_Client-0.20-cp27-none-any.whl
|   |-- py-1.4.31-py2.py3-none-any.whl
|   |-- pytest-2.9.0-py2.py3-none-any.whl
|   |-- selenium-2.52.0-cp27-none-any.whl
|   |-- wheel-0.26.0-py2.py3-none-any.whl
```

Per ulteriori informazioni, consulta [Test 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, troverai la **tests** directory all'interno della directory di lavoro.

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|   |-- test_unittest.py
|-- wheelhouse (directory)
|   |-- Appium_Python_Client-0.20-cp27-none-any.whl
|   |-- py-1.4.31-py2.py3-none-any.whl
|   |-- pytest-2.9.0-py2.py3-none-any.whl
|   |-- selenium-2.52.0-cp27-none-any.whl
|   |-- wheel-0.26.0-py2.py3-none-any.whl
```

Per ulteriori informazioni, consulta [Test 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, troverai la **tests** directory all'interno della directory di lavoro. Il nome del file può essere diverso, ma dovrebbe iniziare con **test_** o terminare con **_test.py**

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|   |-- test_unittest.py
|-- wheelhouse (directory)
|   |-- Appium_Python_Client-0.20-cp27-none-any.whl
|   |-- py-1.4.31-py2.py3-none-any.whl
|   |-- pytest-2.9.0-py2.py3-none-any.whl
|   |-- selenium-2.52.0-cp27-none-any.whl
|   |-- wheel-0.26.0-py2.py3-none-any.whl
```

Per ulteriori informazioni, consulta [Test 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, troverai il *requirements.txt* file all'interno della directory di lavoro.

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|   |-- test_unittest.py
|-- wheelhouse (directory)
|   |-- Appium_Python_Client-0.20-cp27-none-any.whl
|   |-- py-1.4.31-py2.py3-none-any.whl
|   |-- pytest-2.9.0-py2.py3-none-any.whl
|   |-- selenium-2.52.0-cp27-none-any.whl
|   |-- wheel-0.26.0-py2.py3-none-any.whl
```

Per ulteriori informazioni, consulta [Test 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 .
```

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 [Test Appium e AWS Device Farm](#).

APPIUM_WEB_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS

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

Warning

Impossibile installare le wheel di dipendenza. Decomprimere il pacchetto di test, quindi aprire il file requirements.txt e la directory wheelhouse, verificare che le wheel di dipendenza specificate nel file requirements.txt corrispondano esattamente alle wheel di dipendenza all'interno della directory wheelhouse e riprovare.

Si consiglia vivamente di configurare un [virtualenv Python](#) per la creazione dei pacchetti di test. Ecco un esempio di flusso di creazione e attivazione di un ambiente virtuale utilizzando virtualenv Python:

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

Verificare che sia possibile decomprimere il pacchetto di test senza errori. In questo esempio, il nome del pacchetto è test_bundle.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip test_bundle.zip
```

2. Per eseguire test sull'installazione dei file wheel, eseguire il comando seguente:

```
$ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./requirements.txt
```

Un pacchetto Appium Python valido dovrebbe produrre un output come il seguente:

```
Ignoring indexes: https://pypi.python.org/simple
Collecting Appium-Python-Client==0.20 (from -r ./requirements.txt (line 1))
Collecting py==1.4.31 (from -r ./requirements.txt (line 2))
Collecting pytest==2.9.0 (from -r ./requirements.txt (line 3))
Collecting selenium==2.52.0 (from -r ./requirements.txt (line 4))
Collecting wheel==0.26.0 (from -r ./requirements.txt (line 5))
Installing collected packages: selenium, Appium-Python-Client, py, pytest, wheel
  Found existing installation: wheel 0.29.0
    Uninstalling wheel-0.29.0:
      Successfully uninstalled wheel-0.29.0
Successfully installed Appium-Python-Client-0.20 py-1.4.31 pytest-2.9.0
selenium-2.52.0 wheel-0.26.0
```

3. Per disattivare l'ambiente virtuale, eseguire il comando seguente:

```
$ deactivate
```

Per ulteriori informazioni, consulta [Test 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/zhenan/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 [Test 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.

Note

Per considerazioni importanti sull'utilizzo dei test di strumentazione in AWS Device Farm, consulta [Strumentazione per Android e AWS Device Farm](#)

INSTRUMENTATION_TEST_PACKAGE_UNZIP_FAILED

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

```
Warning: We could not open your test APK file. Please verify that the file is valid and try again.
```

Verificare che sia possibile decomprimere il pacchetto di test senza errori. Nell'esempio seguente, il nome del pacchetto è Test-unaligned.apk. app-debug-android

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip app-debug-androidTest-unaligned.apk
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Un pacchetto di test Instrumentation valido produce un output come il seguente:

```
.
|-- AndroidManifest.xml
|-- classes.dex
|-- resources.arsc
|-- LICENSE-junit.txt
|-- junit (directory)
```

```
`-- META-INF (directory)
```

Per ulteriori informazioni, consulta [Strumentazione per Android e AWS Device Farm](#).

INSTRUMENTATION_TEST_PACKAGE_AAPT_DEBUG_BADGING_FAILED

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

```
We could not extract information about your test package. Please verify that the
    test package is valid by running the command "aapt debug badging <path to your
test
    package>", and try again after the command does not print any error.
```

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

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

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

- Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ aapt debug badging app-debug-androidTest-unaligned.apk
```

Un pacchetto di test Instrumentation valido produce un output come il seguente:

```
package: name='com.amazon.aws.adf.android.referenceapp.test' versionCode=''
    versionName='' platformBuildVersionName='5.1.1-1819727'
sdkVersion:'9'
targetSdkVersion:'22'
application-label:'Test-api'
application: label='Test-api' icon=''
application-debuggable
uses-library:'android.test.runner'
feature-group: label=''
uses-feature: name='android.hardware.touchscreen'
uses-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 [Strumentazione per Android e AWS Device Farm](#).

INSTRUMENTATION_TEST_PACKAGE_INSTRUMENTATION_RUNNER_VALU

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

```
We could not find the instrumentation runner value in the AndroidManifest.xml.  
Please verify the test package is valid by running the command "aapt dump xmltree  
<path to  
your test package> AndroidManifest.xml", and try again after finding the  
instrumentation  
runner value behind the keyword "instrumentation."
```

Durante il processo di convalida del caricamento, Device Farm analizza il valore dell'instrumentation runner dall'albero di analisi XML per un file XML contenuto nel pacchetto. Utilizzare il seguente comando: `aapt dump xmltree <path to your package> AndroidManifest.xml`.

Controllare che sia possibile eseguire questo comando nel pacchetto di test Instrumentation e sia possibile trovare il valore instrumentation.

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

- Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ aapt dump xmltree app-debug-androidTest-unaligned.apk AndroidManifest.xml | grep  
-A5 "instrumentation"
```

Un pacchetto di test Instrumentation valido produce un output come il seguente:

```
E: instrumentation (line=9)  
  A: android:label(0x01010001)="Tests for  
com.amazon.aws.adf.android.referenceapp" (Raw: "Tests for  
com.amazon.aws.adf.android.referenceapp")  
  A:  
  android:name(0x01010003)="android.support.test.runner.AndroidJUnitRunner" (Raw:  
"android.support.test.runner.AndroidJUnitRunner")
```



```
A:
android:targetPackage(0x01010021)="com.amazon.aws.adf.android.referenceapp" (Raw:
"com.amazon.aws.adf.android.referenceapp")
A: android:handleProfiling(0x01010022)=(type 0x12)0x0
A: android:functionalTest(0x01010023)=(type 0x12)0x0
```

Per ulteriori informazioni, consulta [Strumentazione per Android e AWS Device Farm](#).

INSTRUMENTATION_TEST_PACKAGE_AAPT_DUMP_XMLTREE_FAILED

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

```
We could not find the valid AndroidManifest.xml in your test package. Please
verify that the test package is valid by running the command "aapt dump xmltree
<path to
your test package> AndroidManifest.xml", and try again after the command does not
print any
error.
```

Durante il processo di convalida del caricamento, Device Farm analizza le informazioni dall'albero di analisi XML per un file XML contenuto nel pacchetto utilizzando il seguente comando: `aapt dump xmltree <path to your package> AndroidManifest.xml`

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

Nel seguente esempio, il nome del pacchetto è Test-unaligned.apk. app-debug-android

- Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ aapt dump xmltree app-debug-androidTest-unaligned.apk AndroidManifest.xml
```

Un pacchetto di test Instrumentation valido produce un output come il seguente:

```
N: android=http://schemas.android.com/apk/res/android
E: manifest (line=2)
A: package="com.amazon.aws.adf.android.referenceapp.test" (Raw:
"com.amazon.aws.adf.android.referenceapp.test")
A: platformBuildVersionCode=(type 0x10)0x16 (Raw: "22")
A: platformBuildVersionName="5.1.1-1819727" (Raw: "5.1.1-1819727")
```

```

E: uses-sdk (line=5)
  A: android:minSdkVersion(0x0101020c)=(type 0x10)0x9
  A: android:targetSdkVersion(0x01010270)=(type 0x10)0x16
E: instrumentation (line=9)
  A: android:label(0x01010001)="Tests for
com.amazon.aws.adf.android.referenceapp" (Raw: "Tests for
com.amazon.aws.adf.android.referenceapp")
  A:
android:name(0x01010003)="android.support.test.runner.AndroidJUnitRunner" (Raw:
"android.support.test.runner.AndroidJUnitRunner")
  A:
android:targetPackage(0x01010021)="com.amazon.aws.adf.android.referenceapp" (Raw:
"com.amazon.aws.adf.android.referenceapp")
  A: android:handleProfiling(0x01010022)=(type 0x12)0x0
  A: android:functionalTest(0x01010023)=(type 0x12)0x0
E: application (line=16)
  A: android:label(0x01010001)=@0x7f020000
  A: android:debuggable(0x0101000f)=(type 0x12)0xffffffff
E: uses-library (line=17)
  A: android:name(0x01010003)="android.test.runner" (Raw:
"android.test.runner")

```

Per ulteriori informazioni, consulta [Strumentazione per Android e AWS Device Farm](#).

INSTRUMENTATION_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSING

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

```

We could not find the package name in your test package. Please verify that the
test package is valid by running the command "aapt debug badging <path to your
test
package>", and try again after finding the package name value behind the keyword
"package:
name."

```

Durante il processo di convalida del caricamento, Device Farm analizza il valore del nome del pacchetto dall'output del seguente comando: `aapt debug badging <path to your package>`

Controllare che sia possibile eseguire questo comando nel pacchetto di test Instrumentation e che sia possibile trovare il valore del nome del pacchetto.

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

- Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ aapt debug badging app-debug-androidTest-unaligned.apk | grep "package: name="
```

Un pacchetto di test Instrumentation valido produce un output come il seguente:

```
package: name='com.amazon.aws.adf.android.referenceapp.test' versionCode=''  
versionName='' platformBuildVersionName='5.1.1-1819727'
```

Per ulteriori informazioni, consulta [Strumentazione per Android e AWS Device Farm](#).

Risoluzione dei problemi relativi ai test delle applicazioni iOS in AWS Device Farm

Il seguente argomento elenca i messaggi di errore che si verificano durante il caricamento dei test di applicazioni iOS e consiglia soluzioni alternative per risolvere ogni errore.

Note

Le seguenti istruzioni si basano su Linux x86_64 e Mac.

IOS_APP_UNZIP_FAILED

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

Warning

Impossibile aprire l'applicazione. Verificare che il file sia valido e riprovare.

Verificare che sia possibile decomprimere il pacchetto dell'applicazione senza errori. Nell'esempio seguente, il nome del pacchetto è AWSDeviceFarm OSReference App.ipa.

1. Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

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

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

Per ulteriori informazioni, consulta [Test iOS in AWS Device Farm](#).

IOS_APP_PAYLOAD_DIR_MISSING

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

Warning

Impossibile trovare la directory Payload nell'applicazione. Decomprimere l'applicazione, verificare che la directory Payload sia nel pacchetto e riprovare.

Nell'esempio seguente, il nome del pacchetto è Farmi App.IPA. AWSDevice OSReference

1. Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Se il pacchetto dell'applicazione iOS è valido, troverai la *Payload* directory all'interno della directory di lavoro.

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

Per ulteriori informazioni, consulta [Test iOS in AWS Device Farm](#).

IOS_APP_APP_DIR_MISSING

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

Warning

Impossibile trovare la directory .app all'interno della directory Payload. Decomprimere l'applicazione, quindi aprire la directory Payload, verificare che la directory .app si trovi all'interno della directory e riprovare.

Nell'esempio seguente, il nome del pacchetto è AWSDeviceFarm iOSReference App.IPA.

1. Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Se il pacchetto dell'applicazione iOS è valido, all'interno della *.app* directory troverai una directory come *AWSDeviceFarmiOSReferenceApp.app* nel nostro esempio. *Payload*

```
.
```

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

Per ulteriori informazioni, consulta [Test iOS in AWS Device Farm](#).

IOS_APP_PLIST_FILE_MISSING

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

Warning

Impossibile trovare il file Info.plist all'interno della directory .app. Decomprimere l'applicazione, quindi aprire la directory .app, verificare che il file Info.plist si trovi all'interno della directory e riprovare.

Nell'esempio seguente, il nome del pacchetto è AWSDeviceFarm iOSReference App.ipa.

1. Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Se il pacchetto dell'applicazione iOS è valido, troverai il *Info.plist* file all'interno della *.app* directory come *AWSDeviceFarmiOSReferenceApp.app* nel nostro esempio.

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

Per ulteriori informazioni, consulta [Test iOS in AWS Device Farm](#).

IOS_APP_CPU_ARCHITECTURE_VALUE_MISSING

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

Warning

Impossibile trovare il valore dell'architettura CPU nel file Info.plist. Decomprimi l'applicazione, quindi apri il file Info.plist nella directory.app, verifica che sia specificata la chiave "UIRequiredDeviceCapabilities" e riprova.

Nell'esempio seguente, il nome del pacchetto è Farmi App.IPA. AWSDevice OSReference

1. Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *AWSDeviceFarmiOSReferenceApp.app* nel nostro esempio:

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

3. Per individuare il valore dell'architettura CPU, aprire Info.plist utilizzando Xcode o Python.

Per Python, installare il modulo biplist eseguendo il seguente comando:

```
$ pip install biplist
```

4. Aprire quindi Python ed eseguire il seguente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['UIRequiredDeviceCapabilities']
```

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

```
['armv7']
```

Per ulteriori informazioni, consulta [Test iOS in AWS Device Farm](#).

IOS_APP_PLATFORM_VALUE_MISSING

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

Warning

Impossibile trovare il valore della piattaforma nel file Info.plist. Decomprimi l'applicazione e quindi apri il file Info.plist nella directory.app, verifica che sia specificata la chiave "CFBundleSupportedPlatforms" e riprova.

Nell'esempio seguente, il nome del pacchetto è Farmi App.IPA. AWSDevice OSReference

1. Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```


Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *AWSDeviceFarmiOSReferenceApp.app* nel nostro esempio:

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

3. Per individuare il valore della piattaforma, aprire Info.plist utilizzando Xcode o Python.

Per Python, installare il modulo biplist eseguendo il seguente comando:

```
$ pip install biplist
```

4. Aprire quindi Python ed eseguire il seguente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['CFBundleSupportedPlatforms']
```

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

```
['iPhoneOS']
```

Per ulteriori informazioni, consulta [Test iOS in AWS Device Farm](#).

IOS_APP_WRONG_PLATFORM_DEVICE_VALUE

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

Warning

Trovato un valore per il dispositivo della piattaforma errato nel file Info.plist. Decomprimi l'applicazione e poi apri il file Info.plist nella directory.app, verifica che il valore della chiave "CFBundleSupportedPlatforms" non contenga la parola chiave «simulator» e riprova.

Nell'esempio seguente, il nome del pacchetto è Farmi App.IPA. AWSDevice OSReference

1. Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *AWSDeviceFarmiOSReferenceApp.app* nel nostro esempio:

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

3. Per individuare il valore della piattaforma, aprire Info.plist utilizzando Xcode o Python.

Per Python, installare il modulo biplist eseguendo il seguente comando:

```
$ pip install biplist
```

4. Aprire quindi Python ed eseguire il seguente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['CFBundleSupportedPlatforms']
```

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

```
['iPhoneOS']
```

Se l'applicazione iOS è valida, il valore non deve contenere la parola chiave: `simulator`.

Per ulteriori informazioni, consulta [Test iOS in AWS Device Farm](#).

IOS_APP_FORM_FACTOR_VALUE_MISSING

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

Warning

Impossibile trovare il valore del fattore di forma nel file Info.plist. Decomprimi l'applicazione e quindi apri il file Info.plist nella directory.app, verifica che sia specificata la chiave "UIDeviceFamiglia» e riprova.

Nell'esempio seguente, il nome del pacchetto è Farmi App.IPA. AWSDevice OSReference

1. Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *AWSDeviceFarmiOSReferenceApp.app* nel nostro esempio:

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

3. Per individuare il valore del fattore di forma, aprire Info.plist utilizzando Xcode o Python.

Per Python, installare il modulo biplist eseguendo il seguente comando:

```
$ pip install biplist
```

4. Aprire quindi Python ed eseguire il seguente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['UIDeviceFamily']
```

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

```
[1, 2]
```

Per ulteriori informazioni, consulta [Test iOS in AWS Device Farm](#).

IOS_APP_PACKAGE_NAME_VALUE_MISSING

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

Warning

Impossibile trovare il valore del nome del pacchetto nel file Info.plist. Decomprimi l'applicazione e quindi apri il file Info.plist nella directory.app, verifica che sia specificata la chiave "CFBundleIdentifier» e riprova.

Nell'esempio seguente, il nome del pacchetto è Farmi App.IPA. AWSDevice OSReference

1. Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *AWSDeviceFarmiOSReferenceApp.app* nel nostro esempio:

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

3. Per individuare il valore del nome del pacchetto, aprire Info.plist utilizzando Xcode o Python.

Per Python, installare il modulo biplist eseguendo il seguente comando:

```
$ pip install biplist
```

4. Aprire quindi Python ed eseguire il seguente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['CFBundleIdentifier']
```

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

```
Amazon.AWSDeviceFarmiOSReferenceApp
```

Per ulteriori informazioni, consulta [Test iOS in AWS Device Farm](#).

IOS_APP_EXECUTABLE_VALUE_MISSING

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

Warning

Impossibile trovare il valore dell'eseguibile nel file Info.plist. Decomprimi l'applicazione e quindi apri il file Info.plist nella directory.app, verifica che sia specificata la chiave "CFBundleExecutable» e riprova.

Nell'esempio seguente, il nome del pacchetto è Farmi App.IPA. AWSDevice OSReference

1. Copiare il pacchetto dell'applicazione nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *AWSDeviceFarmiOSReferenceApp.app* nel nostro esempio:

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

3. Per individuare il valore dell'eseguibile, aprire Info.plist utilizzando Xcode o Python.

Per Python, installare il modulo biplist eseguendo il seguente comando:

```
$ pip install biplist
```

4. Aprire quindi Python ed eseguire il seguente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['CFBundleExecutable']
```

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

```
AWSDeviceFarmiOSReferenceApp
```

Per ulteriori informazioni, consulta [Test iOS in AWS Device Farm](#).

XCTest Test di risoluzione dei problemi in AWS Device Farm

L'argomento seguente elenca i messaggi di errore che si verificano durante il caricamento dei XCTest test e consiglia soluzioni alternative per risolvere ogni errore.

Note

Le istruzioni seguenti presumono l'utilizzo di un sistema MacOS.

XCTEST_TEST_PACKAGE_UNZIP_FAILED

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

Warning

Impossibile aprire il file ZIP del test. Verificare che il file sia valido e riprovare.

Verificare che sia possibile decomprimere il pacchetto dell'applicazione senza errori. Nell'esempio seguente, il nome del pacchetto è `.xctest-1.zip`. `swiftExampleTests`

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Un XCTest pacchetto valido dovrebbe produrre un output simile al seguente:

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

Per ulteriori informazioni, consulta [Integrazione di Device Farm con XCTest iOS](#).

XCTEST_TEST_PACKAGE_XCTEST_DIR_MISSING

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

Warning

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

Nell'esempio seguente, il nome del pacchetto è `swiftExampleTests.xctest-1.zip`.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Se il XCTest pacchetto è valido, troverai una directory con un nome simile a quello contenuto nella directory di *swiftExampleTests.xctest* lavoro. Il nome dovrebbe terminare con *.xctest*.

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

Per ulteriori informazioni, consulta [Integrazione di Device Farm con XCTest iOS](#).

XCTEST_TEST_PACKAGE_PLIST_FILE_MISSING

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

 Warning

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

Nell'esempio seguente, il nome del pacchetto è swiftExampleTests.xctest-1.zip.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

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

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

Per ulteriori informazioni, consulta [Integrazione di Device Farm con XCTest iOS](#).

XCTEST_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING

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

 Warning

Impossibile trovare il valore del nome del pacchetto nel file Info.plist. Decomprimi il pacchetto di test, quindi apri il file Info.plist, verifica che sia specificata la chiave "CFBundleIdentifier» e riprova.

Nell'esempio seguente, il nome del pacchetto è `.xctest-1.zip`. `swiftExampleTests`

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

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

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

3. Per individuare il valore del nome del pacchetto, aprire Info.plist utilizzando Xcode o Python.

Per Python, installare il modulo biplist eseguendo il seguente comando:

```
$ pip install biplist
```

4. Aprire quindi Python ed eseguire il seguente comando:

```
import biplist
info_plist = biplist.readPlist('swiftExampleTests.xctest/Info.plist')
print info_plist['CFBundleIdentifier']
```

Un pacchetto XCtest applicativo valido dovrebbe produrre un output simile al seguente:

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

Per ulteriori informazioni, consulta [Integrazione di Device Farm con XCTest iOS](#).

XCTEST_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING

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

Warning

Impossibile trovare il valore dell'eseguibile nel file Info.plist. Decomprimi il pacchetto di test, quindi apri il file Info.plist, verifica che sia specificata la chiave "CFBundleEseguibile» e riprova.

Nell'esempio seguente, il nome del pacchetto è `.xctest-1.zip`. `swiftExampleTests`

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

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

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

3. Per individuare il valore del nome del pacchetto, aprire Info.plist utilizzando Xcode o Python.

Per Python, installare il modulo biplist eseguendo il seguente comando:

```
$ pip install biplist
```

4. Aprire quindi Python ed eseguire il seguente comando:

```
import biplist  
info_plist = biplist.readPlist('swiftExampleTests.xctest/Info.plist')
```

```
print info_plist['CFBundleExecutable']
```

Un pacchetto XCTest applicativo valido dovrebbe produrre un output simile al seguente:

```
swiftExampleTests
```

Per ulteriori informazioni, consulta [Integrazione di Device Farm con XCTest iOS](#).

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

Il seguente argomento elenca i messaggi di errore che si verificano durante il caricamento dei test dell' XCTest interfaccia utente e consiglia soluzioni alternative per risolvere ogni errore.

Note

Le seguenti istruzioni si basano su Linux x86_64 e Mac.

XCTEST_UI_TEST_PACKAGE_UNZIP_FAILED

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

```
We could not open your test IPA file. Please verify that the file is valid and try again.
```

Verificare che sia possibile decomprimere il pacchetto dell'applicazione senza errori. Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swift-sample-UI.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

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

```
.
|-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
        |   |-- swift-sampleUITests.xctest (directory)
        |       |-- Info.plist
        |       |-- (any other files)
        |-- (any other files)
```

Per ulteriori informazioni, consulta [Integrazione dell' XCTest interfaccia utente per iOS con Device Farm](#).

XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_MISSING

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

We could not find the Payload directory inside your test package. Please unzip your test package, verify that the Payload directory is inside the package, and try again.

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swift-sample-UI.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Se il pacchetto XCTest UI è valido, troverai la **Payload** directory all'interno della directory di lavoro.

```
.
|-- Payload (directory)
```

```
`-- swift-sampleUITests-Runner.app (directory)
    |-- Info.plist
    |-- Plugins (directory)
    |   `-- swift-sampleUITests.xctest (directory)
    |       |-- Info.plist
    |       `-- (any other files)
    `-- (any other files)
```

Per ulteriori informazioni, consulta [Integrazione dell' XCTest interfaccia utente per iOS con Device Farm](#).

XCTEST_UI_TEST_PACKAGE_APP_DIR_MISSING

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

We could not find the .app directory inside the Payload directory. Please unzip your test package and then open the Payload directory, verify that the .app directory is inside the directory, and try again.

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swift-sample-UI.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Se il pacchetto XCTest UI è valido, all'interno della **.app** directory troverai una directory come **swift-sampleUITests-Runner.app** nel nostro esempio. **Payload**

```
.
|-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
        |   `-- swift-sampleUITests.xctest (directory)
        |       |-- Info.plist
        |       `-- (any other files)
        `-- (any other files)
```

```
|
|-- (any other files)
|-- (any other files)
```

Per ulteriori informazioni, consulta [Integrazione dell' XCTest interfaccia utente per iOS con Device Farm](#).

XCTEST_UI_TEST_PACKAGE_PLUGINS_DIR_MISSING

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

We could not find the Plugins directory inside the .app directory. Please unzip your test package and then open the .app directory, verify that the Plugins directory is inside the directory, and try again.

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swift-sample-UI.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Se il pacchetto XCTest UI è valido, troverai la *Plugins* directory all'interno di una *.app* directory. Nel nostro esempio, viene chiamata la directory *swift-sampleUITests-Runner.app*.

```
.
|-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
            |-- swift-sampleUITests.xctest (directory)
                |-- Info.plist
                |-- (any other files)
            |-- (any other files)
```

Per ulteriori informazioni, consulta [Integrazione dell' XCTest interfaccia utente per iOS con Device Farm](#).

XCTEST_UI_TEST_PACKAGE_XCTEST_DIR_MISSING_IN_PLUGINS_DIR

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

We could not find the `.xctest` directory inside the plugins directory. Please unzip your test package and then open the plugins directory, verify that the `.xctest` directory is inside the directory, and try again.

Nel seguente esempio, il nome del pacchetto è `swift-sample-UI.ipa`.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swift-sample-UI.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Se il pacchetto XCTest UI è valido, troverai una `.xctest` directory all'interno della *Plugins* directory. Nel nostro esempio, viene chiamata la directory `swift-sampleUITests.xctest`.

```
.
|-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
            |-- swift-sampleUITests.xctest (directory)
                |-- Info.plist
                |-- (any other files)
            |-- (any other files)
```

Per ulteriori informazioni, consulta [Integrazione dell' XCTest interfaccia utente per iOS con Device Farm](#).

XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING

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

We could not find the Info.plist file inside the .app directory. Please unzip your test package and then open the .app directory, verify that the Info.plist file is inside the directory, and try again.

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swift-sample-UI.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Se il pacchetto XCTest UI è valido, troverai il *Info.plist* file all'interno della *.app* directory. Nel nostro esempio seguente, viene chiamata la directory *swift-sampleUITests-Runner.app*.

```
.
|-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
        |   |-- swift-sampleUITests.xctest (directory)
        |       |-- Info.plist
        |       |-- (any other files)
        |-- (any other files)
```

Per ulteriori informazioni, consulta [Integrazione dell' XCTest interfaccia utente per iOS con Device Farm](#).

XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING_IN_XCTEST_DIR

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

We could not find the Info.plist file inside the .xctest directory. Please unzip your test package and then open the .xctest directory, verify that the Info.plist file is inside the directory, and try again.

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swift-sample-UI.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Se il pacchetto XCTest UI è valido, troverai il *Info.plist* file all'interno della *.xctest* directory. Nel nostro esempio seguente, viene chiamata la directory *swift-sampleUITests.xctest*.

```
.
|-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
        |   |-- swift-sampleUITests.xctest (directory)
        |       |-- Info.plist
        |       |-- (any other files)
        |-- (any other files)
```

Per ulteriori informazioni, consulta [Integrazione dell' XCTest interfaccia utente per iOS con Device Farm](#).

XCTEST_UI_TEST_PACKAGE_CPU_ARCHITECTURE_VALUE_MISSING

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

We could not the CPU architecture value in the Info.plist file. Please unzip your test package and then open the Info.plist file inside the .app

directory, verify that the key "UIRequiredDeviceCapabilities" is specified, and try again.

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swift-sample-UI.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *swift-sampleUITests-Runner.app* nel nostro esempio:

```
.
|-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
        |   |-- swift-sampleUITests.xctest (directory)
        |       |-- Info.plist
        |       |-- (any other files)
        |-- (any other files)
```

3. Per individuare il valore dell'architettura CPU, aprire Info.plist utilizzando Xcode o Python.

Per Python, installare il modulo biplist eseguendo il seguente comando:

```
$ pip install biplist
```

4. Aprire quindi Python ed eseguire il seguente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/
Info.plist')
print info_plist['UIRequiredDeviceCapabilities']
```

Un pacchetto XCTest UI valido dovrebbe produrre un output simile al seguente:

```
['armv7']
```

Per ulteriori informazioni, consulta [Integrazione dell' XCTest interfaccia utente per iOS con Device Farm](#).

XCTEST_UI_TEST_PACKAGE_PLATFORM_VALUE_MISSING

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

We could not find the platform value in the Info.plist. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the key "CFBundleSupportedPlatforms" is specified, and try again.

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swift-sample-UI.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *swift-sampleUITests-Runner.app* nel nostro esempio:

```
.
|-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
            |-- swift-sampleUITests.xctest (directory)
                |-- Info.plist
                |-- (any other files)
            |-- (any other files)
```

3. Per individuare il valore della piattaforma, aprire Info.plist utilizzando Xcode o Python.

Per Python, installare il modulo biplist eseguendo il seguente comando:

```
$ pip install biplist
```

4. Aprire quindi Python ed eseguire il seguente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['CFBundleSupportedPlatforms']
```

Un pacchetto XCTest UI valido dovrebbe produrre un output simile al seguente:

```
['iPhoneOS']
```

Per ulteriori informazioni, consulta [Integrazione dell' XCTest interfaccia utente per iOS con Device Farm](#).

XCTEST_UI_TEST_PACKAGE_WRONG_PLATFORM_DEVICE_VALUE

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

We found the platform device value was wrong in the Info.plist file. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the value of the key "CFBundleSupportedPlatforms" does not contain the keyword "simulator", and try again.

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swift-sample-UI.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *swift-sampleUITests-Runner.app* nel nostro esempio:

```
.
|-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
        |   |-- swift-sampleUITests.xctest (directory)
        |       |-- Info.plist
        |       |-- (any other files)
        |-- (any other files)
```

3. Per individuare il valore della piattaforma, aprire Info.plist utilizzando Xcode o Python.

Per Python, installare il modulo biplist eseguendo il seguente comando:

```
$ pip install biplist
```

4. Aprire quindi Python ed eseguire il seguente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['CFBundleSupportedPlatforms']
```

Un pacchetto XCTest UI valido dovrebbe produrre un output simile al seguente:

```
['iPhoneOS']
```

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

Per ulteriori informazioni, consulta [Integrazione dell' XCTest interfaccia utente per iOS con Device Farm](#).

XCTEST_UI_TEST_PACKAGE_FORM_FACTOR_VALUE_MISSING

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

We could not the form factor value in the Info.plist. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the key "UIDeviceFamily" is specified, and try again.

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swift-sample-UI.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *swift-sampleUITests-Runner.app* nel nostro esempio:

```
.
|-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
            |-- swift-sampleUITests.xctest (directory)
                |-- Info.plist
                |-- (any other files)
            |-- (any other files)
```

3. Per individuare il valore del fattore di forma, aprire Info.plist utilizzando Xcode o Python.

Per Python, installare il modulo biplist eseguendo il seguente comando:

```
$ pip install biplist
```

4. Aprire quindi Python ed eseguire il seguente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['UIDeviceFamily']
```

Un pacchetto XCTest UI valido dovrebbe produrre un output simile al seguente:

```
[1, 2]
```

Per ulteriori informazioni, consulta [Integrazione dell' XCTest interfaccia utente per iOS con Device Farm](#).

XCTEST_UI_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING

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

We could not find the package name value in the Info.plist file. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the key "CFBundleIdentifier" is specified, and try again.

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swift-sample-UI.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *swift-sampleUITests-Runner.app* nel nostro esempio:

```
.
|-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
        |   |-- swift-sampleUITests.xctest (directory)
        |       |-- Info.plist
        |       |-- (any other files)
        |-- (any other files)
```

3. Per individuare il valore del nome del pacchetto, aprire Info.plist utilizzando Xcode o Python.

Per Python, installare il modulo biplist eseguendo il seguente comando:

```
$ pip install biplist
```

4. Aprire quindi Python ed eseguire il seguente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['CFBundleIdentifier']
```

Un pacchetto XCTest UI valido dovrebbe produrre un output simile al seguente:

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

Per ulteriori informazioni, consulta [Integrazione dell' XCTest interfaccia utente per iOS con Device Farm](#).

XCTEST_UI_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING

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

We could not find the executable value in the Info.plist file. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the key "CFBundleExecutable" is specified, and try again.

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swift-sample-UI.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *swift-sampleUITests-Runner.app* nel nostro esempio:

```
.
|-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
        |   |-- swift-sampleUITests.xctest (directory)
        |       |-- Info.plist
        |       |-- (any other files)
        |-- (any other files)
```

3. Per individuare il valore dell'eseguibile, aprire Info.plist utilizzando Xcode o Python.

Per Python, installare il modulo biplist eseguendo il seguente comando:

```
$ pip install biplist
```

4. Aprire quindi Python ed eseguire il seguente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['CFBundleExecutable']
```

Un pacchetto XCTest UI valido dovrebbe produrre un output simile al seguente:

```
XCTRunner
```

Per ulteriori informazioni, consulta [Integrazione dell' XCTest interfaccia utente per iOS con Device Farm](#).

XCTEST_UI_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSING

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

We could not find the package name value in the Info.plist file inside the .xctest directory. Please unzip your test package and then open

the `Info.plist` file inside the `.xctest` directory, verify that the key `"CFBundleIdentifier"` is specified, and try again.

Nel seguente esempio, il nome del pacchetto è `swift-sample-UI.ipa`.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swift-sample-UI.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *swift-sampleUITests-Runner.app* nel nostro esempio:

```
.
|-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
            |-- swift-sampleUITests.xctest (directory)
                |-- Info.plist
                |-- (any other files)
            |-- (any other files)
```

3. Per individuare il valore del nome del pacchetto, aprire `Info.plist` utilizzando Xcode o Python.

Per Python, installare il modulo `biplist` eseguendo il seguente comando:

```
$ pip install biplist
```

4. Aprire quindi Python ed eseguire il seguente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Plugins/
swift-sampleUITests.xctest/Info.plist')
print info_plist['CFBundleIdentifier']
```

Un pacchetto XCtest UI valido dovrebbe produrre un output simile al seguente:

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

Per ulteriori informazioni, consulta [Integrazione dell' XCTest interfaccia utente per iOS con Device Farm](#).

XCTEST_UI_TEST_PACKAGE_TEST_EXECUTABLE_VALUE_MISSING

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

We could not find the executable value in the Info.plist file inside the .xctest directory. Please unzip your test package and then open the Info.plist file inside the .xctest directory, verify that the key "CFBundleExecutable" is specified, and try again.

Nel seguente esempio, il nome del pacchetto è swift-sample-UI.ipa.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swift-sample-UI.ipa
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Dovresti trovare il *Info.plist* file all'interno di una *.app* directory come *swift-sampleUITests-Runner.app* nel nostro esempio:

```
.
|-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
            |-- swift-sampleUITests.xctest (directory)
                |-- Info.plist
                |-- (any other files)
            |-- (any other files)
```

3. Per individuare il valore dell'eseguibile, aprire Info.plist utilizzando Xcode o Python.

Per Python, installare il modulo biplist eseguendo il seguente comando:

```
$ pip install biplist
```

4. Aprire quindi Python ed eseguire il seguente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Plugins/
swift-sampleUITests.xctest/Info.plist')
print info_plist['CFBundleExecutable']
```

Un pacchetto XCTest UI valido dovrebbe produrre un output simile al seguente:

```
swift-sampleUITests
```

Per ulteriori informazioni, consulta [Integrazione dell' XCTest interfaccia utente per iOS con Device Farm](#).

XCTEST_UI_TEST_PACKAGE_MULTIPLE_APP_DIRS

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

We found multiple .app directories inside your test package. Please unzip your test package, verify that only a single .app directory is present inside the package, then try again.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swift-sample-UI.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Se il pacchetto XCTest UI è valido, dovresti trovare solo una singola directory come nel nostro esempio all'interno del pacchetto di test.zip. .app swift-sampleUITests-Runner.app

```

.
|--swift-sample-UI.zip--(directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
        |       `--swift-sampleUITests.xctest (directory)
        |           |-- Info.plist
        |           |-- (any other files)
        |-- (any other files)
    |-- (any other files)

```

Per ulteriori informazioni, consulta [Integrazione dell' XCTest interfaccia utente per iOS con Device Farm](#).

XCTEST_UI_TEST_PACKAGE_MULTIPLE_IPA_DIRS

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

We found multiple .ipa directories inside your test package. Please unzip your test package, verify that only a single .ipa directory is present inside the package, then try again.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swift-sample-UI.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Se il pacchetto XCTest UI è valido, dovresti trovare solo una singola directory come nel nostro esempio all'interno del pacchetto di test.zip. .ipa sampleUITests.ipa

```

.
|--swift-sample-UI.zip--(directory)
    |-- sampleUITests.ipa (directory)
        |-- Payload (directory)
        |-- swift-sampleUITests-Runner.app (directory)

```

```
`-- (any other files)
```

Per ulteriori informazioni, consulta [Integrazione dell' XCTest interfaccia utente per iOS con Device Farm](#).

XCTEST_UI_TEST_PACKAGE_BOTH_APP_AND_IPA_DIR_PRESENT

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

We found both .app and .ipa files inside your test package. Please unzip your test package, verify that only a single .app or .ipa file is present inside the package, then try again.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swift-sample-UI.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Se il pacchetto UI è valido, dovresti trovare una directory simile o una directory come nel nostro esempio all'interno del pacchetto di test.zip. XCTest .ipa sampleUITests.ipa .app swift-sampleUITests-Runner.app Puoi fare riferimento a un esempio di pacchetto di test XCTEST_UI valido nella nostra documentazione su. [Integrazione dell' XCTest interfaccia utente per iOS con Device Farm](#)

```
.
|--swift-sample-UI.zip--(directory)
  |-- sampleUITests.ipa (directory)
    |-- Payload (directory)
      |-- swift-sampleUITests-Runner.app (directory)
    |-- (any other files)
```

oppure

```
.
|--swift-sample-UI.zip--(directory)
```

```
`-- swift-sampleUITests-Runner.app (directory)
    |-- Info.plist
    |-- Plugins (directory)
    |-- (any other files)
`-- (any other files)
```

Per ulteriori informazioni, consulta [Integrazione dell' XCTest interfaccia utente per iOS con Device Farm](#).

XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_PRESENT_IN_ZIP

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

We found a Payload directory inside your .zip test package. Please unzip your test package, ensure that a Payload directory is not present in the package, then try again.

1. Copiare il pacchetto di test nella directory di lavoro, quindi eseguire il comando seguente:

```
$ unzip swift-sample-UI.zip
```

2. Dopo aver decompresso il pacchetto, è possibile trovare la struttura ad albero della directory di lavoro eseguendo il seguente comando:

```
$ tree .
```

Se il pacchetto UI è valido, non dovresti trovare una Payload Directory all'interno del pacchetto di test. XCTest

```
.
|--swift-sample-UI.zip--(directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
        |-- (any other files)
    |-- Payload (directory) [This directory should not be present]
        |-- (any other files)
    |-- (any other files)
```


Per ulteriori informazioni, consulta [Integrazione dell' XCTest interfaccia utente per iOS con Device Farm](#).

Sicurezza in AWS Device Farm

La sicurezza del cloud AWS è la massima priorità. In qualità di AWS cliente, puoi beneficiare di un data center e di un'architettura di rete progettati per soddisfare i requisiti delle organizzazioni più sensibili alla sicurezza.

La sicurezza è una responsabilità condivisa tra AWS e te. Il [modello di responsabilità condivisa](#) descrive questo aspetto come sicurezza del cloud e sicurezza nel cloud:

- **Sicurezza del cloud:** AWS è responsabile della protezione dell'infrastruttura che gestisce AWS i servizi nel AWS cloud. AWS ti fornisce anche servizi che puoi utilizzare in modo sicuro. I revisori esterni testano e verificano regolarmente l'efficacia della nostra sicurezza nell'ambito dei [AWS Programmi di AWS conformità dei Programmi di conformità](#) dei di . Per ulteriori informazioni sui programmi di conformità applicabili AWS Device Farm, consulta [AWS Services in Scope by Compliance Program](#) .
- **Sicurezza nel cloud:** la 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. Imparerai anche a usare altri servizi AWS che ti aiutano a monitorare e proteggere le tue risorse Device Farm.

Argomenti

- [Gestione delle identità e degli accessi in AWS Device Farm](#)
- [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 utilizzi 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 di identità e accesso ad 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. Devi inviare le richieste all'amministratore IAM per cambiare le autorizzazioni degli utenti del servizio. Esamina le informazioni contenute in questa pagina per comprendere i concetti di base relativi a IAM. Per saperne di più su come la tua azienda può utilizzare IAM con Device Farm, consulta [Come funziona AWS Device Farm con IAM](#).

Amministratore IAM: se sei un amministratore IAM, potresti voler conoscere i dettagli su come scrivere policy per gestire l'accesso a Device Farm. Per visualizzare esempi di policy basate sull'identità di Device Farm che puoi utilizzare in IAM, consulta [Esempi di policy basate sull'identità di AWS Device Farm](#).

Autenticazione con identità

L'autenticazione è il modo in cui accedi AWS utilizzando le tue credenziali di identità. Devi essere autenticato (aver effettuato l' Utente root dell'account AWS accesso AWS) come utente IAM o assumendo un ruolo IAM.

Puoi 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. Se accedi come identità federata, l'amministratore ha configurato in precedenza la federazione delle identità utilizzando i ruoli IAM. Quando accedi AWS utilizzando la federazione, assumi 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 (CLI) per firmare crittograficamente le tue richieste utilizzando le tue credenziali. Se non utilizzi AWS strumenti, devi firmare tu stesso le richieste. Per ulteriori informazioni sul metodo consigliato per la firma delle richieste, consulta [Signature Version 4 AWS per le richieste API](#) nella Guida per l'utente IAM.

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'utente di AWS IAM Identity Center e [Utilizzo dell'autenticazione a più fattori \(MFA\)AWS in IAM](#) nella Guida per l'utente IAM.

Account AWS utente root

Quando si crea un account Account AWS, si inizia con un'identità di accesso che ha accesso completo a tutte Servizi AWS 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 un elenco completo delle attività che richiedono l'accesso come utente root, consulta la sezione [Attività che richiedono le credenziali dell'utente root](#) nella Guida per l'utente IAM.

Utenti IAM e gruppi

Un [utente IAM](#) è un'identità interna Account AWS che dispone di autorizzazioni specifiche per una singola persona o applicazione. Ove possibile, consigliamo di fare affidamento a credenziali temporanee invece di creare utenti IAM con credenziali a lungo termine come le password e le chiavi di accesso. Tuttavia, se si hanno casi d'uso specifici che richiedono credenziali a lungo termine con utenti IAM, si consiglia di ruotare le chiavi di accesso. Per ulteriori informazioni, consulta la pagina [Rotazione periodica delle chiavi di accesso per casi d'uso che richiedono credenziali a lungo termine](#) nella Guida per l'utente IAM.

Un [gruppo IAM](#) 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, potresti avere un gruppo denominato IAMAdminse concedere a quel gruppo le autorizzazioni per amministrare le risorse IAM.

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 [Casi d'uso per utenti IAM](#) nella Guida per l'utente IAM.

Ruoli IAM

Un [ruolo IAM](#) è un'identità interna all'utente Account AWS che dispone di autorizzazioni specifiche. È simile a un utente IAM, ma non è associato a una persona specifica. Per assumere temporaneamente un ruolo IAM in AWS Management Console, puoi [passare da un ruolo utente a un ruolo IAM \(console\)](#). Puoi assumere un ruolo chiamando un'operazione AWS CLI o AWS API o utilizzando un URL personalizzato. Per ulteriori informazioni sui metodi per l'utilizzo dei ruoli, consulta [Utilizzo di ruoli IAM](#) nella Guida per l'utente IAM.

I ruoli IAM 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 ulteriori informazioni sulla federazione dei ruoli, consulta [Create a role for a third-party identity provider \(federation\)](#) nella Guida per l'utente IAM. Se utilizzi IAM Identity Center, configura un set di autorizzazioni. IAM Identity Center mette in correlazione il set di autorizzazioni con un ruolo in IAM per controllare a cosa possono accedere le identità dopo l'autenticazione. Per informazioni sui set di autorizzazioni, consulta [Set di autorizzazioni](#) nella Guida per l'utente di AWS IAM Identity Center.
- **Autorizzazioni utente IAM temporanee:** un utente IAM o un ruolo può assumere un ruolo IAM per ottenere temporaneamente autorizzazioni diverse per un'attività specifica.
- **Accesso multi-account:** è possibile utilizzare un ruolo IAM per permettere a un utente (un principale affidabile) con un account diverso di accedere alle risorse nell'account. I ruoli sono lo strumento principale per concedere l'accesso multi-account. Tuttavia, con alcuni Servizi AWS, è possibile allegare una policy direttamente a una risorsa (anziché utilizzare un ruolo come proxy). Per informazioni sulle differenze tra ruoli e policy basate su risorse per l'accesso multi-account, consulta [Accesso a risorse multi-account in IAM](#) nella Guida per l'utente IAM.

- **Accesso a più servizi:** alcuni Servizi AWS utilizzano le funzionalità di altri Servizi AWS. 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 inoltrato (FAS):** quando utilizzi un utente o un ruolo IAM 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 Servizio AWS, combinate con la richiesta Servizio AWS per effettuare richieste ai servizi downstream. Le richieste FAS vengono effettuate solo quando un servizio riceve una richiesta che richiede interazioni con altri Servizi AWS o risorse per essere completata. In questo caso è necessario disporre delle autorizzazioni per eseguire entrambe le operazioni. Per i dettagli delle policy relative alle richieste FAS, consulta [Forward access sessions](#).
- **Ruolo di servizio:** un ruolo di servizio è un [ruolo IAM](#) che un servizio assume per eseguire operazioni per tuo conto. Un amministratore IAM può creare, modificare ed eliminare un ruolo di servizio dall'interno di IAM. Per ulteriori informazioni, consulta la sezione [Create a role to delegate permissions to an Servizio AWS](#) nella Guida per l'utente IAM.
- **Ruolo collegato al servizio:** un ruolo collegato al servizio è un tipo di ruolo di servizio collegato a un Servizio AWS. 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 amministratore IAM può visualizzare le autorizzazioni per i ruoli collegati ai servizi, ma non modificarle.
- **Applicazioni in esecuzione su Amazon EC2:** puoi utilizzare un ruolo IAM per gestire le credenziali temporanee per le applicazioni in esecuzione su un' EC2 istanza e che AWS CLI effettuano richieste AWS API. Questa soluzione è preferibile alla memorizzazione delle 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, crea 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 [Utilizzare un ruolo IAM per concedere le autorizzazioni alle applicazioni in esecuzione su EC2 istanze Amazon](#) nella IAM User Guide.

Come funziona AWS Device Farm con IAM

Prima di utilizzare IAM per gestire l'accesso a Device Farm, è necessario comprendere quali funzionalità IAM sono disponibili per l'uso con Device Farm. Per avere una visione di alto livello di come Device Farm e altri AWS servizi funzionano con IAM, consulta [AWS Services That Work with IAM nella IAM User Guide](#).

Argomenti

- [Politiche basate sull'identità di Device Farm](#)
- [Politiche basate sulle risorse di Device Farm](#)
- [Liste di controllo accessi](#)
- [Autorizzazione basata sui tag Device Farm](#)
- [Ruoli IAM di Device Farm](#)

Politiche basate sull'identità di Device Farm

Con le policy IAM basate su identità, puoi specificare operazioni e risorse consentite o rifiutate, nonché le condizioni in base alle quali le operazioni sono consentite o rifiutate. Device Farm supporta azioni, risorse e chiavi di condizione specifiche. Per informazioni su tutti gli elementi utilizzati in una policy JSON, consulta [Documentazione di riferimento degli elementi delle policy JSON IAM](#) nella Guida per l'utente IAM.

Operazioni

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

L'elemento `Action` di una policy JSON descrive le operazioni che è possibile utilizzare per consentire o negare l'accesso a un criterio. Le azioni politiche in genere hanno lo stesso nome dell'operazione AWS API associata. Ci sono alcune eccezioni, ad esempio le operazioni di sola autorizzazione che non hanno un'operazione API corrispondente. 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'operazione dell'`CreateTestGridUrl` API di test del browser desktop Device Farm, includi

l'elemento `devicefarm:CreateTestGridUrl` nella politica. Le istruzioni della policy devono includere un elemento `Action` o `NotAction`. Device Farm definisce il proprio set di azioni che descrivono le attività che è possibile eseguire con questo servizio.

Per specificare più azioni in una sola istruzione, separa ciascuna di esse con una virgola come mostrato di seguito:

```
"Action": [
    "devicefarm:action1",
    "devicefarm:action2"
```

È possibile specificare più azioni tramite caratteri jolly (*). Ad esempio, per specificare tutte le azioni che iniziano con la parola `List`, includi la seguente azione:

```
"Action": "devicefarm:List*"
```

Per visualizzare un elenco delle azioni di Device Farm, consulta [Azioni definite da AWS Device Farm](#) nello IAM Service Authorization Reference.

Risorse

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

L'elemento JSON `Resource` della policy specifica l'oggetto o gli oggetti ai quali si applica l'operazione. Le istruzioni devono includere un elemento `Resource` o un elemento `NotResource`. Come best practice, specifica una risorsa utilizzando il suo [nome della risorsa Amazon \(ARN\)](#). È possibile eseguire questa operazione per operazioni che supportano un tipo di risorsa specifico, note come autorizzazioni a livello di risorsa.

Per le operazioni 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' EC2 istanza Amazon ha il seguente ARN:

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


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

Ad esempio, per specificare l'istanza `i-1234567890abcdef0` nell'istruzione, utilizza il seguente ARN:

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:instance/i-1234567890abcdef0"
```

Per specificare tutti le istanze che appartengono ad un account specifico, utilizza il carattere jolly (*):

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:instance/*"
```

Alcune azioni di Device Farm, come quelle per la creazione di risorse, non possono essere eseguite su una risorsa. In questi casi, è necessario utilizzare il carattere jolly (*).

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

Molte azioni EC2 dell'API Amazon coinvolgono più risorse. Ad esempio, `AttachVolume` collega un volume Amazon EBS a un'istanza, pertanto un utente IAM dovrà disporre delle autorizzazioni per utilizzare il volume e l'istanza. Per specificare più risorse in una singola istruzione, separale ARNs con virgole.

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

Per visualizzare un elenco dei tipi di risorse Device Farm e relativi ARNs, consulta [Tipi di risorse definiti da AWS Device Farm](#) nello IAM Service Authorization Reference. Per sapere con quali azioni è possibile specificare l'ARN di ogni risorsa, consulta [Actions defined by AWS Device Farm](#) nello IAM Service Authorization Reference.

Chiavi di condizione

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale principale può eseguire operazioni 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 `Condition` è facoltativo. È possibile compilare espressioni

condizionali che utilizzano [operatori di condizione](#), ad esempio uguale a o minore di, per soddisfare la condizione nella policy con i valori nella richiesta.

Se specifichi più elementi `Condition` in un'istruzione o più chiavi in un singolo elemento `Condition`, questi vengono valutati da AWS utilizzando un'operazione AND logica. 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.

È possibile anche utilizzare variabili segnaposto quando specifichi le condizioni. Ad esempio, è possibile autorizzare un utente IAM ad accedere a una risorsa solo se è stata taggata con il relativo nome utente IAM. Per ulteriori informazioni, consulta [Elementi delle policy IAM: variabili e tag](#) nella Guida per l'utente di IAM.

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'utente IAM.

Device Farm definisce il proprio set di chiavi di condizione e supporta anche l'uso di alcune chiavi di condizione globali. Per visualizzare tutte le chiavi di condizione AWS globali, consulta [AWS Global Condition Context Keys](#) nella IAM User Guide.

Per visualizzare un elenco delle chiavi di condizione di Device Farm, consulta [Condition keys for AWS Device Farm](#) nello IAM Service Authorization Reference. Per sapere con quali azioni e risorse puoi utilizzare una chiave di condizione, consulta [Azioni definite da AWS Device Farm](#) nello IAM Service Authorization Reference.

Esempi

Per visualizzare esempi di policy basate sull'identità di Device Farm, vedere. [Esempi di policy basate sull'identità di AWS Device Farm](#)

Politiche basate sulle risorse di Device Farm

Device Farm non supporta politiche basate sulle risorse.

Liste di controllo accessi

Device Farm non supporta gli elenchi di controllo degli accessi (ACLs).

Autorizzazione basata sui tag Device Farm

È possibile allegare tag alle risorse di Device Farm o passare i tag in una richiesta a Device Farm. Per controllare l'accesso basato su tag, 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 [Inserimento di tag in Device Farm](#).

Per visualizzare una policy basata sulle identità di esempio per limitare l'accesso a una risorsa basata su tag su tale risorsa, consulta [Visualizzazione dei progetti di test del browser desktop Device Farm in base ai tag](#).

Ruoli IAM di Device Farm

Un [ruolo IAM](#) è un'entità nel tuo AWS account che dispone di autorizzazioni specifiche.

Utilizzo di credenziali temporanee con Device Farm

Device Farm supporta l'uso di credenziali temporanee.

È possibile utilizzare credenziali temporanee per accedere con la federazione e assumere un ruolo IAM o un ruolo tra account. Puoi ottenere credenziali di sicurezza temporanee chiamando operazioni AWS STS API come o. [AssumeRoleGetFederationToken](#)

Ruoli collegati ai servizi

[I ruoli collegati ai](#) AWS servizi consentono ai servizi di accedere alle risorse di altri servizi per completare un'azione per conto dell'utente. I ruoli collegati ai servizi sono visualizzati nell'account IAM e sono di proprietà del servizio. Un amministratore IAM può visualizzare, ma non modificare, le autorizzazioni per i ruoli collegati ai servizi.

Device Farm utilizza ruoli collegati ai servizi nella funzionalità di test del browser desktop Device Farm. Per informazioni su questi ruoli, consulta [Using Service-Linked Roles in Device Farm Desktop Browser Testing](#) nella guida per sviluppatori.

Ruoli di servizio

Device Farm non supporta i ruoli di servizio.

Questa 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 dei servizi sono visualizzati nell'account IAM e sono di proprietà dell'account. Ciò significa che un amministratore IAM può modificare le autorizzazioni per questo ruolo. Tuttavia, questo potrebbe pregiudicare la funzionalità del servizio.

Gestione dell'accesso 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 documenti JSON. Per ulteriori informazioni sulla struttura e sui contenuti dei documenti delle policy JSON, consulta [Panoramica delle policy JSON](#) nella Guida per l'utente IAM.

Gli amministratori possono utilizzare le policy AWS JSON per specificare chi ha accesso a cosa. In altre parole, quale principale può eseguire operazioni 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 operazioni sulle risorse di cui hanno bisogno, un amministratore IAM può creare policy IAM. L'amministratore può quindi aggiungere le policy IAM ai ruoli e gli utenti possono assumere i ruoli.

Le policy IAM definiscono le autorizzazioni relative a un'operazione, a prescindere dal metodo utilizzato per eseguirla. Ad esempio, supponiamo di disporre di una policy che consente l'operazione `iam:GetRole`. Un utente con tale policy può ottenere informazioni sul ruolo dall' AWS Management Console AWS CLI, dall'API AWS API.

Policy basate sull'identità

Le policy basate su identità sono documenti di policy di autorizzazione JSON che è possibile allegare a un'identità (utente, gruppo di utenti o ruolo IAM). Tali policy definiscono le operazioni che utenti e ruoli possono eseguire, su quali risorse e in quali condizioni. Per informazioni su come creare una policy basata su identità, consulta [Definizione di autorizzazioni personalizzate IAM con policy gestite dal cliente](#) nella Guida per l'utente 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 nel tuo Account AWS.

Le politiche gestite includono politiche AWS gestite e politiche gestite dai clienti. Per informazioni su come scegliere tra una policy gestita o una policy inline, consulta [Scelta fra policy gestite e policy inline](#) nella Guida per l'utente IAM.

La tabella seguente descrive le policy gestite da Device Farm in AWS.

Modifica	Descrizione	Data
AWSDeviceFarmFullAccess	Fornisce accesso completo a tutte le operazioni di AWS Device Farm.	15 luglio 2015
AWSServiceRoleForDeviceFarmTestGrid	Consente a Device Farm di accedere alle risorse AWS per tuo conto.	20 maggio 2021

Altri tipi di policy

AWS supporta tipi di policy aggiuntivi 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 delle autorizzazioni è una funzionalità avanzata nella quale si imposta il numero massimo di autorizzazioni che una policy basata su identità può concedere a un'entità IAM (utente o ruolo 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 IAM](#) nella Guida per l'utente IAM.
- **Politiche di controllo del servizio (SCP):** SCPs sono politiche JSON 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 di proprietà dell'Account AWS azienda. Se abiliti tutte le funzionalità di un'organizzazione, puoi applicare le politiche di controllo del servizio (SCP) a uno o tutti i tuoi account. L'SCP limita le autorizzazioni per le entità presenti negli account dei membri, inclusa ciascuna di esse. Utente root dell'account AWS. Per ulteriori informazioni su Organizations and SCPs, consulta [le politiche di controllo dei servizi](#) nella Guida AWS Organizations per l'utente.

- Politiche di controllo delle risorse (RCPs): RCPs sono politiche JSON che puoi utilizzare per impostare le autorizzazioni massime disponibili per le risorse nei tuoi account senza aggiornare le politiche IAM allegate a ciascuna risorsa di tua proprietà. L'RCP limita le autorizzazioni per le risorse negli account dei membri e può influire sulle autorizzazioni effettive per le identità, incluse le Utente root dell'account AWS, indipendentemente dal fatto che appartengano o meno all'organizzazione. Per ulteriori informazioni su Organizations e RCPs, incluso un elenco di Servizi AWS tale supporto RCPs, vedere [Resource control policies \(RCPs\)](#) nella Guida per l'AWS Organizations utente.
- 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 [Policy di sessione](#) nella Guida per l'utente IAM.

Più tipi di policy

Quando più tipi di policy si applicano a una richiesta, le autorizzazioni risultanti sono più complicate da comprendere. Per scoprire come si AWS determina se consentire o meno una richiesta quando sono coinvolti più tipi di policy, consulta la [logica di valutazione delle policy](#) nella IAM User Guide.

Esempi di policy basate sull'identità di AWS Device Farm

Per impostazione predefinita, gli utenti e i ruoli IAM non dispongono dell'autorizzazione per creare o modificare risorse Device Farm. Inoltre, non possono eseguire attività utilizzando l' AWS API AWS Management Console AWS CLI, o. Un amministratore IAM deve creare policy IAM che concedono a utenti e ruoli l'autorizzazione per eseguire operazioni API specifiche sulle risorse specificate di cui hanno bisogno. L'amministratore deve quindi allegare queste policy a utenti o IAM che richiedono tali autorizzazioni.

Per informazioni su come creare una policy basata su identità IAM utilizzando questi documenti di policy JSON di esempio, consulta [Creazione di policy nella scheda JSON](#) nella Guida per l'utente IAM.

Argomenti

- [Best practice delle policy](#)
- [Consentire agli utenti di visualizzare le loro autorizzazioni](#)

- [Accesso a un progetto di test del browser desktop Device Farm](#)
- [Visualizzazione dei progetti di test del browser desktop Device Farm in base ai tag](#)

Best practice delle policy

Le politiche basate sull'identità determinano se qualcuno può creare, accedere o eliminare le risorse Device Farm nel tuo account. Queste operazioni 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 [Policy gestite da AWS](#) o [Policy gestite da AWS per le funzioni dei processi](#) nella Guida per l'utente IAM.
- Applica le autorizzazioni con privilegio minimo: quando imposti le autorizzazioni con le policy IAM, concedi solo le autorizzazioni richieste per eseguire un'attività. È possibile farlo definendo le azioni che possono essere intraprese su risorse specifiche in condizioni specifiche, note anche come autorizzazioni con privilegi minimi. Per ulteriori informazioni sull'utilizzo di IAM per applicare le autorizzazioni, consulta [Policy e autorizzazioni in IAM](#) nella Guida per l'utente IAM.
- Condizioni d'uso nelle policy IAM per limitare ulteriormente l'accesso: per limitare l'accesso a operazioni e risorse è possibile aggiungere una condizione alle tue policy. Ad esempio, è possibile scrivere una condizione di policy per specificare che tutte le richieste devono essere inviate utilizzando SSL. Puoi anche utilizzare le condizioni per concedere l'accesso alle azioni del servizio se vengono utilizzate tramite uno specifico Servizio AWS, ad esempio AWS CloudFormation. Per ulteriori informazioni, consulta la sezione [Elementi delle policy JSON di IAM: condizione](#) nella Guida per l'utente IAM.
- Utilizzo di IAM Access Analyzer per convalidare le policy IAM e garantire autorizzazioni sicure e funzionali: IAM Access Analyzer convalida le policy nuove ed esistenti in modo che aderiscano alla sintassi della policy IAM (JSON) e alle best practice di IAM. IAM Access Analyzer offre oltre 100 controlli delle policy e consigli utili per creare policy sicure e funzionali. Per ulteriori informazioni, consulta [Convalida delle policy per il Sistema di analisi degli accessi IAM](#) nella Guida per l'utente IAM.
- Richiedi l'autenticazione a più fattori (MFA): se hai uno scenario che richiede utenti IAM o un utente root nel Account AWS tuo, attiva l'MFA per una maggiore sicurezza. Per richiedere la MFA

quando vengono chiamate le operazioni API, aggiungi le condizioni MFA alle policy. Per ulteriori informazioni, consulta [Protezione dell'accesso API con MFA](#) nella Guida per l'utente IAM.

Per maggiori informazioni sulle best practice in IAM, consulta [Best practice di sicurezza in IAM](#) nella Guida per l'utente di IAM.

Consentire agli utenti di visualizzare le loro autorizzazioni

Questo esempio mostra in che modo è possibile creare una policy che consente agli utenti IAM di visualizzare le policy inline e gestite che sono collegate alla relativa identità utente. Questa policy include le autorizzazioni per completare questa azione sulla console o utilizzando programmaticamente l'API o. AWS CLI AWS

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsWithUser",
        "iam:ListAttachedUserPolicies",
        "iam:ListUserPolicies",
        "iam:GetUser"
      ],
      "Resource": ["arn:aws:iam::*:user/${aws:username}"]
    },
    {
      "Sid": "NavigateInConsole",
      "Effect": "Allow",
      "Action": [
        "iam:GetGroupPolicy",
        "iam:GetPolicyVersion",
        "iam:GetPolicy",
        "iam:ListAttachedGroupPolicies",
        "iam:ListGroupPolicies",
        "iam:ListPolicyVersions",
        "iam:ListPolicies",
        "iam:ListUsers"
      ],
      "Resource": "*"
    }
  ]
}
```



```
    }
  ]
}
```

Accesso a un progetto di test del browser desktop Device Farm

In questo esempio, vuoi concedere a un utente IAM del tuo AWS account l'accesso a uno dei tuoi progetti di test del browser desktop Device Farm, `arn:aws:devicefarm:us-west-2:111122223333:testgrid-project:123e4567-e89b-12d3-a456-426655441111`. Si desidera che l'account sia in grado di visualizzare gli elementi correlati al progetto.

Oltre all'endpoint `devicefarm:GetTestGridProject`, l'account deve avere gli endpoint `devicefarm:ListTestGridSessions`, `devicefarm:GetTestGridSession`, `devicefarm:ListTestGridSessionActions` e `devicefarm:ListTestGridSessionArtifacts`.

```
{
  "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 policy IAM delinea una policy minima per consentire a un CI runner di avviare un test di un nuovo test dell'app nativa di Device Farm creando un caricamento e utilizzandolo per pianificare un'esecuzione di test:

```
{
  "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 policy agli utenti IAM nel tuo account. Se un utente denominato `richard-roe` tenta di visualizzare un progetto o una sessione di Device Farm, il progetto deve essere taggato `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 la sezione [Elementi delle policy JSON di IAM: condizione](#) nella Guida per l'utente di IAM.

Risoluzione dei problemi di identità e accesso ad AWS Device Farm

Utilizza le seguenti informazioni per aiutarti a diagnosticare e risolvere i problemi più comuni che potresti riscontrare quando lavori con Device Farm e IAM.

Non sono autorizzato a eseguire un'azione in Device Farm

Se ricevi un messaggio di errore 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.

Il seguente errore di esempio si verifica quando l'utente IAM tenta di utilizzare la console per visualizzare i dettagli di un'esecuzione, ma non dispone di autorizzazioni `devicefarm:GetRun` sulle 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'operazione `iam:PassRole`, le tue politiche devono essere aggiornate per consentirti di trasferire un ruolo a Device Farm.

Alcuni Servizi AWS consentono di trasferire un ruolo esistente a quel servizio invece di creare un nuovo ruolo di servizio o un ruolo collegato al servizio. Per eseguire questa operazione, è necessario disporre delle autorizzazioni per trasmettere il ruolo al servizio.

Il seguente errore di esempio si verifica quando un utente IAM denominato `marymajor` tenta di utilizzare la console per eseguire un'azione in Device Farm. Tuttavia, l'azione richiede che il servizio disponga delle autorizzazioni concesse da un ruolo di servizio. Mary non dispone delle autorizzazioni per 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 aiuto, contatta il tuo AWS amministratore. L'amministratore è colui che ti ha fornito le credenziali di accesso.

Desidero visualizzare le mie chiavi di accesso

Dopo aver creato le chiavi di accesso utente IAM, è possibile visualizzare il proprio ID chiave di accesso in qualsiasi momento. Tuttavia, non è possibile visualizzare nuovamente la chiave di accesso segreta. Se perdi la chiave segreta, dovrai creare una nuova coppia di chiavi di accesso.

Le chiavi di accesso sono composte da due parti: un ID chiave di accesso (ad esempio `AKIAIOSFODNN7EXAMPLE`) e una chiave di accesso segreta (ad esempio, `wJa1rXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY`). Come un nome utente e una password, è necessario utilizzare sia l'ID chiave di accesso sia la chiave di accesso segreta insieme per autenticare le richieste dell'utente. Gestisci le tue chiavi di accesso in modo sicuro mentre crei il nome utente e la password.

 Important

Non fornire le chiavi di accesso a terze parti, neppure per aiutare a [trovare l'ID utente canonico](#). In questo modo, potresti concedere a qualcuno l'accesso permanente al tuo Account AWS.

Quando crei una coppia di chiavi di accesso, ti viene chiesto di salvare l'ID chiave di accesso e la chiave di accesso segreta in una posizione sicura. La chiave di accesso segreta è disponibile solo al momento della creazione. Se si perde la chiave di accesso segreta, è necessario aggiungere nuove chiavi di accesso all'utente IAM. È possibile avere massimo due chiavi di accesso. Se se ne hanno già due, è necessario eliminare una coppia di chiavi prima di crearne una nuova. Per visualizzare le istruzioni, consulta [Gestione delle chiavi di accesso](#) nella Guida per l'utente di IAM.

Sono un amministratore e desidero consentire ad altri di accedere a Device Farm

Per consentire ad altri di accedere a Device Farm, è necessario concedere l'autorizzazione alle persone o alle applicazioni che devono accedere. Se si utilizza AWS IAM Identity Center per gestire persone e applicazioni, si assegnano set di autorizzazioni a utenti o gruppi per definire il loro livello di accesso. I set di autorizzazioni creano e assegnano automaticamente le policy IAM ai ruoli IAM associati alla persona o all'applicazione. Per ulteriori informazioni, consulta [Set di autorizzazioni](#) nella Guida per l'AWS IAM Identity Center utente.

Se non utilizzi IAM Identity Center, devi creare entità IAM (utenti o ruoli) per le persone o le applicazioni che necessitano di accesso. È quindi necessario allegare una 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 utenti, gruppi, policy e autorizzazioni IAM, consulta [IAM Identities](#) and [Policies and permissions in IAM nella IAM](#) User Guide.

Desidero consentire a persone esterne al mio AWS account di accedere alle mie risorse di Device Farm

È possibile creare un ruolo con il quale utenti in altri account o persone esterne all'organizzazione possono accedere alle tue risorse. È possibile specificare chi è attendibile per l'assunzione del ruolo. Per i servizi che supportano politiche basate sulle risorse o liste di controllo degli accessi (ACLs), puoi utilizzare tali politiche per concedere alle persone l'accesso alle tue risorse.

Per ulteriori informazioni, consulta gli argomenti seguenti:

- Per sapere se Device Farm supporta queste funzionalità, vedere [Come funziona AWS Device Farm con IAM](#).
- Per scoprire come fornire l'accesso alle tue risorse su tutto Account AWS ciò che possiedi, consulta [Fornire l'accesso a un utente IAM in un altro Account AWS di tua proprietà](#) nella IAM User Guide.
- Per scoprire come fornire l'accesso alle tue risorse a terze parti Account AWS, consulta [Fornire l'accesso a soggetti Account AWS di proprietà di terze parti](#) nella Guida per l'utente IAM.
- Per informazioni su come fornire l'accesso tramite la federazione delle identità, consulta [Fornire l'accesso a utenti autenticati esternamente \(Federazione delle identità\)](#) nella Guida per l'utente IAM.
- Per informazioni sulle differenze di utilizzo tra ruoli e policy basate su risorse per l'accesso multi-account, consulta [Accesso a risorse multi-account in IAM](#) nella Guida per l'utente IAM.

Convalida della conformità per AWS Device Farm

I revisori esterni valutano la sicurezza e la conformità nell' AWS Device Farm ambito di più programmi di AWS conformità. Questi includono SOC, PCI, FedRAMP, HIPAA e altri. AWS Device Farm non rientra nell'ambito di alcun programma di conformità. AWS

Per un elenco di AWS servizi nell'ambito di programmi di conformità specifici, consulta [Servizi AWS nell'ambito del programma di conformità](#) . Per informazioni generali, consulta Programmi di [AWS conformità Programmi](#) di di .

È possibile scaricare report di audit di terze parti utilizzando AWS Artifact. Per ulteriori informazioni, consulta [Download dei report in AWS Artifact](#).

La responsabilità di conformità dell'utente nell'utilizzo di Device Farm è determinata dalla sensibilità dei dati, dagli obiettivi di conformità dell'azienda e dalle leggi e dai regolamenti applicabili. AWS fornisce le seguenti risorse per contribuire alla conformità:

- [Security and Compliance Quick Start Guides \(Guide Quick Start Sicurezza e compliance\)](#): queste guide alla distribuzione illustrano considerazioni relative all'architettura e forniscono procedure per la distribuzione di ambienti di base incentrati sulla sicurezza e sulla conformità su AWS.
- [AWS Risorse per](#) la per la conformità: questa raccolta di cartelle di lavoro e guide potrebbe riguardare il settore e la località in cui operi.

- [Evaluating Resources with Rules](#) nella AWS Config Developer Guide: AWS Config valuta la conformità delle configurazioni delle risorse alle pratiche interne, alle linee guida del settore e alle normative.
- [AWS Security Hub](#)— 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](#) si applica alla protezione dei dati in AWS Device Farm (Device Farm). Come descritto in questo modello, AWS è responsabile della protezione dell'infrastruttura globale che gestisce tutti i Cloud AWS. L'utente è responsabile del controllo dei contenuti ospitati su questa infrastruttura. L'utente è inoltre responsabile della configurazione della protezione e delle attività di gestione per i Servizi AWS utilizzati. Per ulteriori informazioni sulla privacy dei dati, vedi le [Domande frequenti sulla privacy dei dati](#). Per informazioni sulla protezione dei dati in Europa, consulta il post del blog relativo al [Modello di responsabilità condivisa AWS e GDPR](#) nel Blog sulla sicurezza AWS .

Ai fini della protezione dei dati, consigliamo di proteggere Account AWS le credenziali e configurare i singoli utenti con AWS IAM Identity Center or AWS Identity and Access Management (IAM). In tal modo, a ogni utente verranno assegnate solo le autorizzazioni necessarie per svolgere i suoi compiti. 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 le risorse. AWS È richiesto TLS 1.2 ed è consigliato TLS 1.3.
- Configura l'API e la registrazione delle attività degli utenti con. AWS CloudTrail Per informazioni sull'utilizzo dei CloudTrail percorsi per acquisire AWS le attività, consulta [Lavorare con i CloudTrail percorsi](#) nella Guida per l'AWS CloudTrail utente.
- Utilizza soluzioni di AWS crittografia, insieme a tutti i controlli di sicurezza predefiniti all'interno Servizi AWS.
- Utilizza i servizi di sicurezza gestiti avanzati, come Amazon Macie, che aiutano a individuare e proteggere i dati sensibili archiviati in Amazon S3.
- Se hai bisogno di moduli crittografici convalidati FIPS 140-3 per accedere AWS tramite un'interfaccia a riga di comando o un'API, usa un endpoint FIPS. Per ulteriori informazioni sugli endpoint FIPS disponibili, consulta il [Federal Information Processing Standard \(FIPS\) 140-3](#).

Ti consigliamo di non inserire mai informazioni riservate o sensibili, ad esempio gli indirizzi e-mail dei clienti, nei tag o nei campi di testo in formato libero, ad esempio nel campo Nome. Ciò include quando lavori con Device Farm o altro Servizi AWS utilizzando la console AWS CLI, l'API o AWS SDKs. I dati inseriti nei tag o nei campi di testo in formato libero utilizzati per i nomi possono essere utilizzati per la fatturazione o i log di diagnostica. Quando fornisci un URL a un server esterno, ti suggeriamo vivamente di non includere informazioni sulle credenziali nell'URL per convalidare la tua richiesta al server.

Crittografia in transito

Gli endpoint Device Farm supportano solo HTTPS firmato (SSL/TLS) requests except where otherwise noted. All content retrieved from or placed in Amazon S3 through upload URLs is encrypted using SSL/TLS. Per ulteriori informazioni su come vengono effettuate le richieste HTTPS AWS, consulta [Firmare le richieste AWS API](#) nella Guida AWS generale.

La crittografia e la protezione di tutte le comunicazioni effettuate dalle applicazioni testate, nonché di eventuali applicazioni aggiuntive installate durante l'esecuzione di test sul dispositivo, è responsabilità dell'utente.

Crittografia a riposo

La funzionalità di test del browser desktop di Device Farm supporta la crittografia a riposo per gli artefatti generati durante i test.

I dati di test fisici dei dispositivi mobili di Device Farm non sono crittografati quando sono inattivi.

Conservazione dei dati

I dati in Device Farm vengono conservati per un periodo di tempo limitato. Dopo la scadenza del periodo di conservazione, i dati vengono rimossi dall'archivio di backup di Device Farm.

Tipo di contenuto	Periodo di conservazione (giorni)	Periodo di conservazione dei metadati (giorni)
Applicazioni caricate	30	30
Pacchetti di test caricati	30	30
Log	400	400

Tipo di contenuto	Periodo di conservazione (giorni)	Periodo di conservazione dei metadati (giorni)
Registrazioni video e altri artefatti	400	400

È responsabilità dell'utente archiviare qualsiasi contenuto che desidera conservare per periodi più lunghi.

Gestione dei dati

I dati in Device Farm vengono gestiti in modo diverso a seconda delle funzionalità utilizzate. Questa sezione spiega come vengono gestiti i dati durante e dopo l'utilizzo di Device Farm.

Test del browser desktop

Le istanze utilizzate durante le sessioni Selenium non vengono salvate. Tutti i dati generati come risultato delle interazioni del browser vengono scartati al termine della sessione.

Questa funzionalità attualmente supporta la crittografia a riposo per gli artefatti generati durante il test.

Test fisici dei dispositivi

Le seguenti sezioni forniscono informazioni sui passaggi AWS necessari per pulire o distruggere i dispositivi dopo aver utilizzato Device Farm.

I dati di test sui dispositivi mobili fisici di Device Farm non sono crittografati quando sono inattivi.

Flotte di dispositivi pubblici

Una volta completata l'esecuzione del test, Device Farm esegue una serie di attività di pulizia su ogni dispositivo del parco dispositivi pubblici, inclusa la disinstallazione dell'app. Se non siamo in grado di confermare la disinstallazione della tua app o qualsiasi altra fase di pulizia, il dispositivo viene reimpostato ai valori di fabbrica prima di essere riutilizzato.

Note

In alcuni casi è possibile che i dati persistano tra una sessione e l'altra, soprattutto se si utilizza il sistema del dispositivo al di fuori del contesto dell'app. Per questo motivo, e poiché

Device Farm acquisisce video e registri delle attività che si svolgono durante l'utilizzo di ciascun dispositivo, consigliamo di non inserire informazioni sensibili (ad esempio, account Google o ID Apple), informazioni personali e altri dettagli sensibili alla sicurezza durante le sessioni di test automatizzate e di accesso remoto.

Dispositivi privati

Dopo la scadenza o la risoluzione del contratto relativo a un dispositivo privato, quest'ultimo viene ritirato dall'uso ed eliminato in modo sicuro in conformità con le policy di eliminazione di AWS. Per ulteriori informazioni, consulta [Dispositivi privati in AWS Device Farm](#).

Gestione delle chiavi

Attualmente, Device Farm non offre alcuna gestione di chiavi esterne per la crittografia dei dati, a riposo o in transito.

Riservatezza del traffico Internet

Device Farm può essere configurato, solo per dispositivi privati, per utilizzare gli endpoint Amazon VPC per connettersi alle tue risorse. AWS L'accesso a qualsiasi AWS infrastruttura non pubblica associata al tuo account (ad esempio, EC2 istanze Amazon senza un indirizzo IP pubblico) deve utilizzare un endpoint Amazon VPC. Indipendentemente dalla configurazione degli endpoint VPC, Device Farm isola il traffico dagli altri utenti in tutta la rete Device Farm.

Non è garantito che le connessioni all'esterno della AWS rete siano protette o protette, ed è responsabilità dell'utente proteggere tutte le connessioni Internet create dalle applicazioni.

Resilienza in AWS Device Farm

L'infrastruttura AWS globale è costruita attorno a AWS regioni e zone di disponibilità. AWS Le regioni forniscono più zone di disponibilità fisicamente separate e isolate, collegate con reti a bassa latenza, ad alto throughput e altamente ridondanti. Con le zone di disponibilità, puoi progettare e gestire applicazioni e database che eseguono automaticamente il failover tra zone di disponibilità senza interruzioni. Le zone di disponibilità sono più disponibili, tolleranti ai guasti e scalabili rispetto alle infrastrutture a data center singolo o multiplo tradizionali.

[Per ulteriori informazioni su AWS regioni e zone di disponibilità, consulta Global Infrastructure.AWS](#)

Poiché Device Farm è disponibile solo nella us-west-2 regione, si consiglia vivamente di implementare processi di backup e ripristino. Device Farm non dovrebbe essere l'unica fonte di contenuti caricati.

Device Farm non garantisce la disponibilità di dispositivi pubblici. Questi dispositivi vengono inseriti e rimossi dal pool di dispositivi pubblici in base a una varietà di fattori, come il tasso di errore e lo stato di quarantena. Si consiglia di non dipendere dalla disponibilità di un dispositivo nel pool di dispositivi pubblici.

Sicurezza dell'infrastruttura in AWS Device Farm

In quanto servizio gestito, AWS Device Farm è protetto dalla sicurezza della rete AWS globale. Per informazioni sui servizi AWS di sicurezza e su come AWS protegge l'infrastruttura, consulta [AWS Cloud Security](#). Per progettare il tuo AWS ambiente utilizzando le migliori pratiche per la sicurezza dell'infrastruttura, vedi [Infrastructure Protection](#) in Security Pillar AWS Well-Architected Framework.

Si utilizzano chiamate API AWS pubblicate per accedere a Device Farm attraverso la rete. I client devono supportare quanto segue:

- Transport Layer Security (TLS). È richiesto TLS 1.2 ed è consigliato TLS 1.3.
- Suite di cifratura con Perfect Forward Secrecy (PFS), ad esempio Ephemeral Diffie-Hellman (DHE) o Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). La maggior parte dei sistemi moderni, come Java 7 e versioni successive, supporta tali modalità.

Inoltre, le richieste devono essere firmate utilizzando un ID chiave di accesso e una chiave di accesso segreta associata a un principale IAM. 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 endpoint Amazon VPC per proteggere le connessioni in entrata e in uscita dal tuo account. AWS

Sicurezza dell'infrastruttura per il test dei browser desktop

Quando si utilizza la caratteristica di test del browser desktop, tutte le sessioni di test sono separate l'una dall'altra. Le istanze Selenium non possono comunicare tra loro senza una terza parte intermedia, esterna a AWS.

Tutto il traffico verso i WebDriver controller Selenium deve essere effettuato tramite l'endpoint HTTPS generato con `createTestGridUrl`.

L'utente ha la responsabilità di assicurarsi che ogni istanza di test di Device Farm abbia accesso sicuro alle risorse da essa testate. Per impostazione predefinita, le istanze di test dei browser desktop di Device Farm hanno accesso alla rete Internet pubblica. Quando colleghi un'istanza a un VPC, questa si comporta come qualsiasi altra EC2 istanza, con accesso alle risorse determinato dalla configurazione del VPC e dai componenti di rete associati. AWS fornisce [gruppi di sicurezza](#) e [liste di controllo degli accessi di rete \(ACLs\)](#) per aumentare la sicurezza nel tuo VPC. I gruppi di sicurezza controllano il traffico in entrata e in uscita per le tue risorse e la rete ACLs controlla il traffico in entrata e in uscita per le tue sottoreti. I gruppi di sicurezza forniscono un controllo di accesso sufficiente per la maggior parte delle sottoreti. Puoi usare la rete ACLs se desideri un ulteriore livello di sicurezza per il tuo VPC. Per linee guida generali sulle best practice di sicurezza per l'utilizzo di Amazon VPCs, consulta [le best practice di sicurezza](#) per il tuo VPC nella Amazon Virtual Private Cloud User Guide.

Analisi e gestione delle vulnerabilità di configurazione in Device Farm

Device Farm consente di eseguire software che non sono attivamente gestiti o aggiornati dal fornitore, ad esempio il fornitore del sistema operativo, il fornitore dell'hardware o l'operatore telefonico. Device Farm fa del suo meglio per mantenere aggiornato il software, ma non garantisce che una particolare versione del software su un dispositivo fisico sia aggiornata, poiché la progettazione consente l'utilizzo di software potenzialmente vulnerabili.

Ad esempio, se viene eseguito un test su un dispositivo con Android 4.4.2, Device Farm non garantisce che il dispositivo sia aggiornato contro la [vulnerabilità di Android](#) nota come StageFright.

Spetta al fornitore (e talvolta al gestore) del dispositivo fornire aggiornamenti per la sicurezza ai dispositivi. Non è possibile garantire che un'applicazione dannosa che utilizza questa vulnerabilità sia catturata dalla quarantena automatica.

I dispositivi privati vengono mantenuti in base al contratto stipulato con AWS.

Device Farm compie ogni sforzo possibile per impedire alle applicazioni dei clienti di eseguire azioni come il rooting o il jailbreak. Device Farm rimuove i dispositivi messi in quarantena dal pool pubblico fino a quando non vengono esaminati manualmente.

È tua responsabilità mantenere aggiornate tutte le librerie o le versioni del software che usi nei tuoi test, come Python wheels e Ruby gems. Device Farm consiglia di aggiornare le librerie di test.

Queste risorse consentono di mantenere aggiornate le dipendenze dei test:

- Per informazioni su come proteggere le gemme Ruby, consulta [Security Practices](#) sul RubyGems sito web.
- [Per informazioni sul pacchetto di sicurezza utilizzato da Pipenv e approvato dalla Python Packaging Authority per scansione il grafico delle dipendenze alla ricerca di vulnerabilità note, vedere Rilevamento delle vulnerabilità di sicurezza su GitHub](#)
- [Per informazioni sul correttore di dipendenze Maven dell'Open Web Application Security Project \(OWASP\), consulta OWASP sul sito Web OWASP. DependencyCheck](#)

È importante ricordare che anche se un sistema automatizzato non ritiene che ci siano problemi di sicurezza noti, ciò non significa che non ci siano problemi di sicurezza. Utilizzare sempre la due diligence quando si utilizzano librerie o strumenti di terze parti e verificare le firme crittografiche quando possibile o ritenuto necessario.

Risposta agli incidenti in Device Farm

Device Farm monitora continuamente i dispositivi per rilevare comportamenti che potrebbero indicare problemi di sicurezza. Se AWS viene a conoscenza di un caso in cui i dati del cliente, come i risultati dei test o i file scritti su un dispositivo pubblico, sono accessibili da un altro cliente, AWS contatta i clienti interessati, in base alle politiche standard di avviso e segnalazione degli incidenti utilizzate in tutti i servizi AWS.

Registrazione e monitoraggio in Device Farm

Questo servizio supporta AWS CloudTrail, che è un servizio che registra AWS le chiamate Account AWS e invia i file di registro a un bucket Amazon S3. Utilizzando le informazioni raccolte da CloudTrail, è possibile determinare a quali richieste sono state inoltrate correttamente Servizi AWS, chi ha effettuato la richiesta, quando è stata effettuata e così via. Per ulteriori informazioni CloudTrail, incluso come attivarlo e trovare i file di registro, consulta la [Guida per l'AWS CloudTrail utente](#).

Per informazioni sull'utilizzo CloudTrail con Device Farm, vedere [Registrazione delle chiamate API AWS Device Farm con AWS CloudTrail](#).

Best practice di sicurezza per Device Farm

Device Farm offre una serie di funzionalità di sicurezza da prendere in considerazione durante lo sviluppo e l'implementazione delle proprie politiche di sicurezza. Le seguenti best practice sono linee guida generali e non rappresentano una soluzione di sicurezza completa. Poiché queste best practice potrebbero non essere appropriate o sufficienti per l'ambiente, gestiscile come considerazioni utili anziché prescrizioni.

- Concedere a qualsiasi sistema di integrazione continua (CI) utilizzato i privilegi minimi possibili in IAM. Prendere in considerazione l'utilizzo di credenziali temporanee per ogni test di sistema CI in modo che, anche se un sistema CI è compromesso, non può effettuare richieste false. Per ulteriori informazioni sulle credenziali temporanee, consulta la [IAM User Guide](#).
- Utilizzare i comandi adb in un ambiente di test personalizzato per pulire qualsiasi contenuto creato dall'applicazione. Per ulteriori informazioni sugli ambienti di test personalizzati, consulta [Ambienti di test personalizzati](#)

Limiti in AWS Device Farm

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 c'è limite al numero di dispositivi che puoi includere in un'esecuzione di test. Il numero di dispositivi (job) che possono eseguire i test in parallelo in un dato momento è pari alla concorrenza a livello di account. La concorrenza predefinita a livello di account per l'uso misurato in Device Farm è cinque.

Il limite di concorrenza misurato può essere aumentato su richiesta fino a una determinata soglia a seconda del caso d'uso. La concorrenza predefinita a livello di account per un utilizzo illimitato è pari al numero di slot a cui sei abbonato per quella piattaforma.

[Per ulteriori informazioni sui limiti di concorrenza misurati di default o sulle quote in generale, consulta la pagina Quote.](#)

- Device Farm segue un algoritmo token-bucket per limitare la frequenza delle chiamate API. Ad esempio, immagina di creare un bucket che contiene token. Ogni token rappresenta una transazione e una chiamata API utilizza un token. I token vengono aggiunti al bucket a una velocità fissa (ad esempio, 10 token al secondo) e il bucket ha una capacità massima (ad esempio, 100 token). Quando arriva una richiesta o un pacchetto, deve richiedere un token dal bucket per essere elaborato. Se i token sono sufficienti, la richiesta viene autorizzata e i token vengono rimossi. Se non ci sono abbastanza token, la richiesta viene ritardata o abbandonata, a seconda dell'implementazione.

In Device Farm, ecco come viene implementato l'algoritmo:

- Le richieste API Burst sono il numero massimo di richieste a cui il servizio è in grado di rispondere per un'API specificata in un ID di account cliente specificato. In altre parole, questa

è la capacità del bucket. Puoi chiamare l'API tante volte quanti sono i token rimasti nel bucket e ogni richiesta consuma un token.

- La frequenza Transactions-per-second (TPS) è la velocità minima alla quale le richieste API possono essere eseguite. In altre parole, questa è la velocità con cui il bucket si riempie di token al secondo. Ad esempio, se un'API ha un numero burst pari a dieci ma un TPS pari a uno, puoi chiamarla dieci volte all'istante. Tuttavia, il bucket recupererebbe i token solo alla velocità di un token al secondo, con conseguente limitazione a una chiamata al secondo, a meno che tu non smetta di chiamare l'API per lasciare che il bucket si riempia.

Ecco le tariffe di Device Farm APIs:

- Per List and Get APIs, la capacità delle richieste dell'API Burst è 50 e la tariffa Transactions-per-second (TPS) è. 10
- Per tutti gli altri APIs, la capacità delle richieste dell'API Burst è e 10 la velocità Transactions-per-second (TPS) è. 1

Strumenti e plugin per AWS Device Farm

Questa sezione contiene collegamenti e informazioni sull'utilizzo degli strumenti e dei plugin di AWS Device Farm. Puoi trovare i plugin Device Farm su [AWS Labs](#) su GitHub

Se sei uno sviluppatore Android, abbiamo anche un'[app di esempio AWS Device Farm per Android su GitHub](#). Puoi utilizzare l'app e i test di esempio come riferimento per i tuoi script di test di Device Farm.

Argomenti

- [Integrazione di Device Farm con un server CI Jenkins](#)
- [Integrazione di Device Farm con un sistema di build Gradle](#)

Integrazione di Device Farm con un server CI Jenkins

Il plug-in Jenkins CI fornisce la funzionalità AWS Device Farm dal tuo server di integrazione continua (CI) Jenkins. Per ulteriori informazioni, consulta [Jenkins \(software\)](#).

Note

Per scaricare il plugin Jenkins, vai a [GitHub](#) e segui le istruzioni in. [Fase 1: Installazione del plugin Jenkins CI per AWS Device Farm](#)

Questa sezione contiene una serie di procedure per configurare e utilizzare il plugin Jenkins CI con AWS Device Farm.

Le seguenti immagini mostrano le caratteristiche del plugin Jenkins CI.

 **Jenkins**

Jenkins > Hello World App >

 [Back to Dashboard](#)

 [Status](#)

 [Changes](#)

 [Workspace](#)

 [Build Now](#)

 [Delete Project](#)

 [Configure](#)

 [AWS Device Farm](#)

 **Build History**

[trend](#) 

	#19	Jul 15, 2015 4:25 AM
	#18	Jul 15, 2015 1:35 AM
	#17	Jul 15, 2015 1:21 AM
	#16	Jul 15, 2015 1:06 AM
	#15	Jul 14, 2015 10:55 PM

 [RSS for all](#)  [RSS for failures](#)

Project Hello World App

 [Workspace](#)

 [Recent Changes](#)



Recent AWS Device Farm Results

Status	Build Number	Pass/Warn/Skip/Fail/Error/Stop	Web Report
Completed	#19	12✔ 0⚠ 1🔄 1❌ 1! 0■	Full Report
Completed	#18	9✔ 0⚠ 1🔄 1❌ 1! 0■	Full Report
Completed	#17	12✔ 0⚠ 1🔄 1❌ 1! 0■	Full Report
Completed	#16	12✔ 0⚠ 1🔄 1❌ 1! 0■	Full Report
Completed	#15	11✔ 0⚠ 1🔄 2❌ 1! 0■	Full Report

Permalinks

- [Last build \(#19\), 41 min ago](#)
- [Last failed build \(#19\), 41 min ago](#)
- [Last unsuccessful build \(#19\), 41 min ago](#)

Plug-in Jenkins CI

Versione API 2015-06-23 341

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.

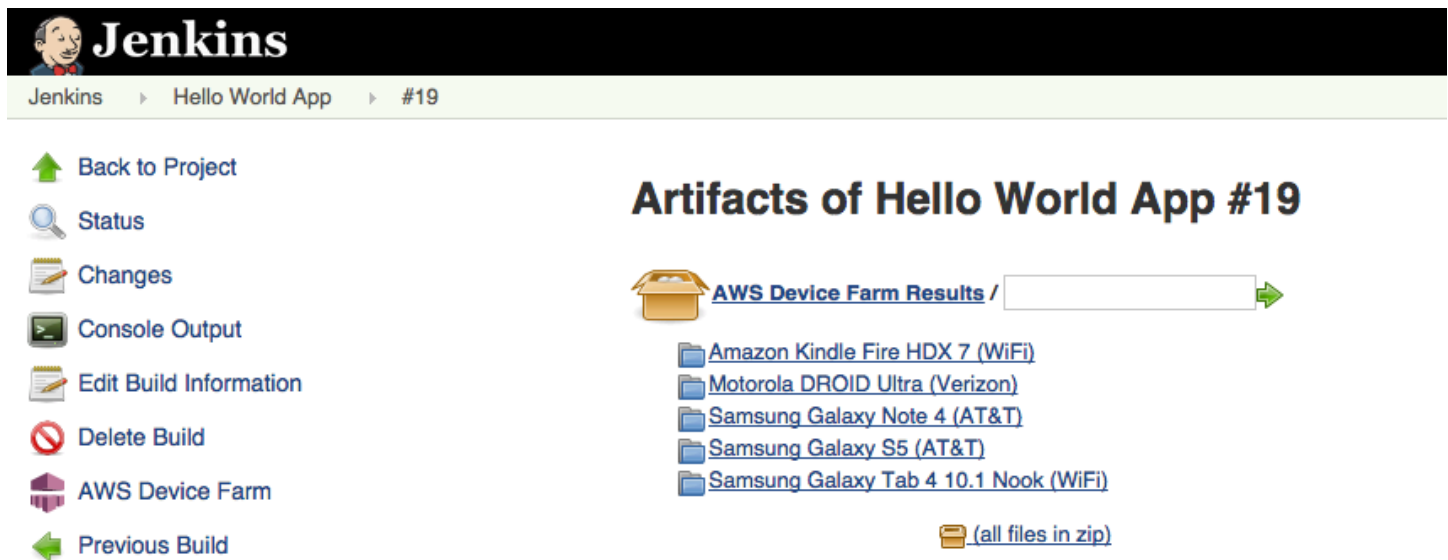
Tags ?

[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

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 \(WiFi\)](#)

[\(all files in zip\)](#)

Argomenti

- [Dipendenze](#)
- [Fase 1: Installazione del plugin Jenkins CI per AWS Device Farm](#)
- [Fase 2: creazione di un AWS Identity and Access Management utente per il plugin Jenkins CI per AWS Device Farm](#)
- [Fase 3: Configurazione del plugin Jenkins CI per la prima volta in AWS Device Farm](#)
- [Fase 4: Usare il plugin in un job Jenkins](#)

Dipendenze

Il plugin Jenkins CI richiede AWS Mobile SDK 1.10.5 o versione successiva. Per ulteriori informazioni e per installare l'SDK, consulta [SDK AWS Mobile](#).

Fase 1: Installazione del plugin Jenkins CI per AWS Device Farm

Esistono due opzioni per installare il plug-in Jenkins continuous integration (CI) per AWS Device Farm. Puoi cercare il plugin nella finestra di dialogo Available Plugins (Plugin disponibili) nell'interfaccia utente Web di Jenkins, oppure puoi scaricare il file `hpi` e installarlo in Jenkins.

Installazione dall'interfaccia utente di Jenkins

1. Trova il plugin nell'interfaccia utente di Jenkins selezionando Manage Jenkins (Gestisci Jenkins), Manage Plugins (Gestisci Plugin) e Available (Disponibili).

2. Cercare aws-device-farm.
3. Installa il plug-in AWS Device Farm.
4. Assicurati che il plugin sia di proprietà dell'utente Jenkins.
5. Riavvia Jenkins.

Scarica il plugin

1. Scaricate il hpi file direttamente da <http://updates.jenkins-ci.org/latest/aws-device-farm.hpi>.
2. Assicurati che il plugin sia di proprietà dell'utente Jenkins.
3. Installa il plugin utilizzando una delle seguenti opzioni.
 - Carica il plugin selezionando Manage Jenkins (Gestisci Jenkins), Manage Plugins (Gestisci plugin), Advanced (Avanzate) e Upload plugin (Carica plugin).
 - Inserisci il file hpi nella directory del plugin Jenkins (di solito /var/lib/jenkins/plugins).
4. Riavvia Jenkins.

Fase 2: creazione di un AWS Identity and Access Management utente per il plugin Jenkins CI per AWS Device Farm

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

Per creare un nuovo utente IAM, consulta [Creazione di un utente IAM \(AWS Management Console\)](#). 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.

Autorizza l'utente IAM ad accedere a Device Farm

Per concedere all'utente IAM l'autorizzazione ad accedere a Device Farm, crea una nuova policy di accesso in IAM, quindi assegna la policy di accesso all'utente IAM come segue.

 Note

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

Per creare la politica di accesso in IAM

1. Aprire la console IAM all'indirizzo <https://console.aws.amazon.com/iam/>.
2. Seleziona Policy.
3. 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 politica di accesso.
4. Nell'area Permissions (Autorizzazioni), in Managed Policies (Policy gestite), seleziona Attach Policy (Collega policy).
5. Seleziona la policy che hai appena creato (ad esempio, AWSDeviceFarmAccessPolicy).
6. Scegli Attach Policy (Collega policy).

Fase 3: Configurazione del plugin Jenkins CI per la prima volta in AWS Device Farm

La prima volta che esegui il server Jenkins, dovrai configurare il sistema nel seguente modo.

Note

Se stai utilizzando [slot per i dispositivi](#), la loro funzionalità sarà disabilitata per impostazione predefinita.

1. Accedi all'interfaccia utente Web di Jenkins.
2. Sul lato sinistro dello schermo, scegli Manage Jenkins (Gestisci Jenkins).
3. Scegli Configure System (Configura sistema).
4. Scorri verso il basso fino all'installazione di AWS Device Farm.
5. Copia le tue credenziali di sicurezza da [Creazione di un utente IAM per il tuo plugin Jenkins CI](#) e incolla il tuo ID chiave di accesso e la tua chiave di accesso segreta nelle rispettive caselle.
6. 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. Fai clic su Aggiungi azione post-compilazione e seleziona Run Tests on AWS Device Farm.
6. Seleziona il progetto che desideri utilizzare.
7. Seleziona il pool di dispositivi che desideri utilizzare.
8. Seleziona se archiviare in locale gli artefatti dei test (ad esempio, i log e gli screenshot).
9. In Application (Applicazione), inserisci il percorso dell'applicazione compilata.
10. Seleziona il test che desideri eseguire e compila tutti i campi obbligatori.
11. Seleziona Salva.

Integrazione di Device Farm con un sistema di build Gradle

Il plugin Device Farm Gradle fornisce l'integrazione di AWS Device Farm con il sistema di build Gradle in Android Studio. Per ulteriori informazioni, consulta [Gradle](#).

Note

Per scaricare il plugin Gradle, vai a [GitHub](#) e segui le istruzioni in [Creazione del plugin Device Farm Gradle](#)

Il plugin Device Farm Gradle fornisce la funzionalità Device Farm dal tuo ambiente Android Studio. Puoi iniziare i test su telefoni e tablet Android reali ospitati da Device Farm.

Questa sezione contiene una serie di procedure per configurare e utilizzare il Device Farm Gradle Plugin.

Argomenti

- [Dipendenze](#)
- [Fase 1: creazione del plug-in AWS Device Farm Gradle](#)
- [Fase 2: configurazione del plugin AWS Device Farm Gradle](#)
- [Fase 3: Generazione di un utente IAM nel plugin Device Farm Gradle](#)
- [Fase 4: Configurazione dei tipi di test](#)

Dipendenze

Runtime

- Il Device Farm Gradle Plugin richiede AWS Mobile SDK 1.10.15 o successivo. Per ulteriori informazioni e per installare l'SDK, consulta [SDK AWS Mobile](#).
- Android Tools Builder Test API 0.5.2
- Apache Commons Lang3 3.3.4

Per unit test

- Testng 6.8.8
- Jmockit 1.19
- Android Gradle Tools 1.3.0

Fase 1: creazione del plug-in AWS Device Farm Gradle

Questo plugin fornisce l'integrazione di AWS Device Farm con il sistema di build Gradle in Android Studio. Per ulteriori informazioni, consulta [Gradle](#).

Note

La creazione del plugin è facoltativa. Il plugin viene pubblicato tramite Maven Central. Se desideri consentire a Gradle di scaricare direttamente il plugin, salta questa fase e passa a [Fase 2: configurazione del plugin AWS Device Farm Gradle](#).

Per creare il plugin

1. Vai al [GitHub](#) repository e clona.
2. Creare il plugin usando `gradle install`.

Il plugin viene installato sull'archivio Maven locale.

Fase successiva: [Fase 2: configurazione del plugin AWS Device Farm Gradle](#)

Fase 2: configurazione del plugin AWS Device Farm Gradle

Se non lo hai già fatto, clona l'archivio e installa il plugin usando la procedura indicata qui: [Creazione del plugin Device Farm Gradle](#).

Per configurare il plugin AWS Device Farm Gradle

1. Aggiungere l'artefatto del plugin al proprio elenco di dipendenze in `build.gradle`.

```
buildscript {  
  
    repositories {  
        mavenLocal()  
        mavenCentral()  
    }  
  
    dependencies {  
        classpath 'com.android.tools.build:gradle:1.3.0'  
        classpath 'com.amazonaws:aws-devicefarm-gradle-plugin:1.0'  
    }  
}
```

2. Configurare il plugin nel proprio file `build.gradle`. La seguente configurazione della specifica di test deve fungere da guida:

```
apply plugin: 'devicefarm'  
  
devicefarm {  
  
    // Required. The project must already exist. You can create a project in the  
    // AWS Device Farm console.  
    projectName "My Project" // required: Must already exist.  
  
    // Optional. Defaults to "Top Devices"  
    // devicePool "My Device Pool Name"  
  
    // Optional. Default is 150 minutes  
    // executionTimeoutMinutes 150  
  
    // Optional. Set to "off" if you want to disable device video recording during  
    // a run. Default is "on"  
    // videoRecording "on"  
  
    // Optional. Set to "off" if you want to disable device performance monitoring  
    // during a run. Default is "on"  
    // performanceMonitoring "on"
```

```

    // Optional. Add this if you have a subscription and want to use your unmetered
    slots
    // useUnmeteredDevices()

    // Required. You must specify either accessKey and secretKey OR roleArn.
    roleArn takes precedence.
    authentication {
        accessKey "AKIAIOSFODNN7EXAMPLE"
        secretKey "wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY"

        // OR

        roleArn "arn:aws:iam::111122223333:role/DeviceFarmRole"
    }

    // Optionally, you can
    // - enable or disable Wi-Fi, Bluetooth, GPS, NFC radios
    // - set the GPS coordinates
    // - specify files and applications that must be on the device when your test
    runs
    devicestate {
        // Extra files to include on the device.
        // extraDataZipFile file("path/to/zip")

        // Other applications that must be installed in addition to yours.
        // auxiliaryApps files(file("path/to/app"), file("path/to/app2"))

        // By default, Wi-Fi, Bluetooth, GPS, and NFC are turned on.
        // wifi "off"
        // bluetooth "off"
        // gps "off"
        // nfc "off"

        // You can specify GPS location. By default, this location is 47.6204,
        -122.3491
        // latitude 44.97005
        // longitude -93.28872
    }

    // By default, the Instrumentation test is used.
    // If you want to use a different test type, configure it here.
    // You can set only one test type (for example, Calabash, Fuzz, and so on)

    // Fuzz

```

```
// fuzz { }  
  
// Calabash  
// calabash { tests file("path-to-features.zip") }  
  
}
```

3. Esegui il test di Device Farm utilizzando la seguente attività:`gradle devicefarmUpload`.

L'output della build stamperà un collegamento alla console Device Farm dove è possibile monitorare l'esecuzione del test.

Fase successiva: [Generazione di un utente IAM nel plugin Device Farm Gradle](#)

Fase 3: Generazione di un utente IAM nel plugin Device Farm Gradle

AWS Identity and Access Management (IAM) ti aiuta a gestire le autorizzazioni e le politiche per lavorare con AWS le risorse. Questo argomento illustra come generare un utente IAM con autorizzazioni per accedere alle risorse di AWS Device Farm.

Se non l'hai già fatto, completa i passaggi 1 e 2 prima di generare un utente IAM.

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

Note

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

Per creare un nuovo utente con la politica di accesso appropriata in IAM

1. Aprire la console IAM all'indirizzo <https://console.aws.amazon.com/iam/>.
2. Scegliere Users (Utenti).
3. Scegliere Create New Users (Crea nuovi utenti).

4. Immettere il nome utente di propria scelta.

Ad esempio **GradleUser**.

5. Scegli Create (Crea) .
6. Scegliere Download Credentials (Scarica credenziali) e salvarle in una posizione in cui saranno facilmente recuperabili in seguito.
7. Scegli Chiudi.
8. Selezionare il nome utente nell'elenco.
9. In Permissions (Autorizzazioni), espandere l'intestazione Inline Policies (Policy inline) facendo clic sulla freccia giù a destra.
10. Scegli Fai clic qui dove dice, Non ci sono politiche in linea da mostrare. Per crearne uno, clicca qui.
11. Sulla schermata Set Permissions (Imposta autorizzazioni), selezionare Custom Policy (Personalizza policy).
12. Scegli Seleziona.
13. Assegnare un nome alla policy, ad esempio **AWSDeviceFarmGradlePolicy**.
14. Incollare la seguente policy in Policy Document (Documento policy).

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

15. Scegli Apply Policy (Applica policy).

Fase successiva: [Configurazione dei tipi di test](#).

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

Fase 4: Configurazione dei tipi di test

Per impostazione predefinita, il plug-in AWS Device Farm Gradle esegue il [Strumentazione per Android e AWS Device Farm](#) test. Se desideri eseguire i tuoi test o specificare parametri aggiuntivi, puoi scegliere di configurare un tipo di test. Questo argomento fornisce informazioni su ciascun tipo di test disponibile e su ciò che devi fare in Android Studio per configurarlo per l'uso. Per ulteriori informazioni sui tipi di test disponibili in Device Farm, vedere [Framework di test e test integrati in AWS Device Farm](#).

Se non l'hai già fatto, completa i passaggi da 1 a 3 prima di configurare i tipi di test.

Note

Se stai utilizzando [slot per i dispositivi](#), la loro funzionalità sarà disabilitata per impostazione predefinita.

Appium

Device Farm fornisce supporto per Appium Java e JUnit TestNG per Android.

- [Appium \(in Java \(\)\) JUnit](#)
- [Appium \(in Java \(TestNg\)\)](#)

Puoi scegliere `useTestNG()` o `useJUnit()`. JUnit è l'impostazione predefinita e non deve essere specificata in modo esplicito.

```
appium {  
    tests file("path to zip file") // required  
    useTestNG() // or useJUnit()  
}
```

Incorporato: fuzz

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

```
fuzz {
```

```
eventThrottle 50 // optional default
eventCount 6000 // optional default
randomizerSeed 1234 // optional default blank

}
```

Per ulteriori informazioni, consulta [Esecuzione del fuzz test integrato di Device Farm \(Android e iOS\)](#).

Instrumentation

Device Farm fornisce supporto per la strumentazione (EspressoJUnit, Robotium o qualsiasi test basato sulla strumentazione) per Android. Per ulteriori informazioni, consulta [Strumentazione per Android e AWS Device Farm](#).

Quando si esegue un test di strumentazione in Gradle, Device Farm utilizza il .apk file generato dalla directory AndroidTest come fonte dei test.

```
instrumentation {

    filter "test filter per developer docs" // optional

}
```

Cronologia dei documenti di AWS Device Farm

La tabella seguente descrive le modifiche importanti apportate alla documentazione dall'ultima versione della guida.

Modifica	Descrizione	Data della modifica
AL2 supporto	Device Farm ora supporta l'ambiente di AL2 test per Android. Ulteriori informazioni su AL2 .	6 novembre 2023
Migrazione da ambienti di test standard a ambienti di test personalizzati	Guida alla migrazione aggiornata all'obsolescenza dei documenti per i test in modalità standard nel dicembre 2023.	3 settembre 2023
Supporto VPC ENI	Device Farm ora consente ai dispositivi privati di utilizzare la funzionalità di connettività VPC-ENI per aiutare i clienti a 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: • Appium (Python) • Appium (Python)	20 aprile 2020
Nuove informazioni sulla sicurezza e informazioni sull'eticheizzazione delle risorse AWS .	Per rendere la protezione dei servizi AWS più semplice e completa, 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 Inserimento di tag 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 plugin Device Farm Gradle .	16 agosto 2019
Nuovo requisito per i test eseguiti con XCTest	Per le esecuzioni di test che utilizzano il XCTest framework, Device Farm ora richiede un pacchetto di app creato per i test. Ulteriori informazioni su the section called "XCTest" .	4 febbraio 2019
Supporto per i tipi di test Appium Node.js e Appium Ruby in ambienti personalizzati	È ora possibile eseguire i test in entrambi gli ambienti di test personalizzati Appium Node.js e Appium Ruby. Ulteriori informazioni su Framework di test e test integrati in AWS Device Farm .	10 gennaio 2019

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 1.7.2, 1.7.1 e 1.6.5. È inoltre possibile eseguire i test con versioni 1.8.1 e 1.8.0 utilizzando un file YAML di specifica di test personalizzato in un ambiente personalizzato. Ulteriori informazioni su Framework di test e test integrati in AWS Device Farm .	2 ottobre 2018
Ambienti di test personalizzati	Con un ambiente di test personalizzato, puoi assicurarti che i test vengano eseguiti come nel tuo ambiente locale. Device Farm ora fornisce supporto per i log live e lo streaming video, in modo da poter ottenere un feedback immediato sui test eseguiti in un ambiente di test personalizzato. Ulteriori informazioni su Ambienti di test personalizzati in AWS Device Farm .	16 agosto 2018
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 Integrazione di AWS Device Farm in una fase di CodePipeline 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 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 Impostazione del timeout di esecuzione per le esecuzioni di test in AWS Device Farm .	9 febbraio 2017
Configurazione di reti	È ora possibile simulare le condizioni e le connessioni di rete per una sessione di test. Ulteriori informazioni su Simulazione delle connessioni e delle condizioni di rete per le esecuzioni di AWS Device Farm .	8 dicembre 2016
Nuova sezione di risoluzione dei problemi	Ora puoi risolvere i problemi di caricamento dei pacchetti di test utilizzando una serie di procedure progettate per risolvere i messaggi di errore che potresti riscontrare nella console Device Farm. Ulteriori informazioni su Risoluzione degli errori di Device Farm .	10 agosto 2016
Sessioni di accesso remoto	È ora possibile accedere da remoto e interagire con un singolo dispositivo nella console. Ulteriori informazioni su Accesso remoto .	19 aprile 2016
Slot di dispositivi self-service	Ora puoi acquistare slot per dispositivi utilizzando AWS Management Console, the o l' AWS Command Line Interface API. Ulteriori informazioni su come Acquisto di uno slot per dispositivi in Device Farm .	22 marzo 2016

Modifica	Descrizione	Data della modifica
Come arrestare sessioni di test	Ora puoi interrompere le esecuzioni dei test utilizzando l'AWS Management Console AWS Command Line Interface , la o l'API. Ulteriori informazioni su come Interruzione di un'esecuzione in AWS Device Farm .	22 marzo 2016
XCTest Nuovi tipi di test dell'interfaccia utente	Ora puoi eseguire test personalizzati XCTest dell'interfaccia utente sulle applicazioni iOS. Ulteriori informazioni sul tipo di test Integrazione dell' XCTest interfaccia utente per iOS con Device Farm .	8 marzo 2016
Nuovi tipi di test per Appium Python	È ora possibile eseguire test personalizzati per Appium Python su Android, iOS e applicazioni Web. Ulteriori informazioni su Framework di test e test integrati in AWS Device Farm .	19 gennaio 2016
Tipi di test per applicazioni Web	Ora puoi eseguire test personalizzati Appium Java e JUnit TestNg su applicazioni web. Ulteriori informazioni su Test di app Web in AWS Device Farm .	19 Novembre 2015
Plugin AWS Device Farm Gradle	Ulteriori informazioni su come installare e utilizzare il Plugin Device Farm Gradle .	28 settembre 2015
Nuovo test integrato per Android: Explorer	Il test Explorer ricerca per indicizzazione nell'app analizzando ogni schermata come se fosse un utente finale e acquisendo screenshot durante l'analisi.	16 settembre 2015
Supporto iOS aggiunto	Scopri di più sul test dei dispositivi iOS e sull'esecuzione di test iOS (inclusi XCTest) in Framework di test e test integrati in AWS Device Farm .	4 agosto 2015
Versione pubblica iniziale	Questa è la versione pubblica iniziale della AWS Device Farm Developer Guide.	13 luglio 2015

AWS Glossario

Per la AWS terminologia più recente, consultate il [AWS glossario](#) nella sezione Reference. Glossario AWS

Le traduzioni sono generate tramite traduzione automatica. In caso di conflitto tra il contenuto di una traduzione e la versione originale in Inglese, quest'ultima prevarrà.