



Entwicklerhandbuch

AWS Device Farm



API-Version 2015-06-23

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

AWS Device Farm: Entwicklerhandbuch

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

Die Marken und Handelsmarken von Amazon dürfen nicht in einer Weise in Verbindung mit nicht von Amazon stammenden Produkten oder Services verwendet werden, die geeignet ist, Kunden irrezuführen oder Amazon in irgendeiner Weise herabzusetzen oder zu diskreditieren. Alle anderen Handelsmarken, die nicht Eigentum von Amazon sind, gehören den jeweiligen Besitzern, die möglicherweise zu Amazon gehören oder nicht, mit Amazon verbunden sind oder von Amazon gesponsert werden.

Table of Contents

Was ist AWS Device Farm?	1
Automatisiertes Testen von Apps	1
Interaktion mit Fernzugriff	1
Terminologie	2
Einrichtung	3
Einrichtung	4
Schritt 1: Melden Sie sich an für AWS	4
Schritt 2: Erstellen oder verwenden Sie einen IAM-Benutzer in Ihrem Konto AWS	4
Schritt 3: Erteilen Sie dem IAM-Benutzer die Erlaubnis, auf Device Farm zuzugreifen	5
Nächster Schritt	6
Erste Schritte	7
Voraussetzungen	7
Schritt 1: Melden Sie sich bei der -Konsole an	8
Schritt 2: Erstellen Sie ein Projekt	8
Schritt 3: Einen Lauf erstellen und starten	8
Schritt 4: Sehen Sie sich die Ergebnisse des Testlaufs an	10
Nächste Schritte	11
Erwerben Sie Geräteslots	12
Geräteslots (Konsole) kaufen	12
Erwerben Sie einen Geräteslot (AWS CLI)	14
Erwerben Sie einen Geräteslot (API)	18
Stornieren Sie einen Geräteslot (Konsole)	19
Einen Geräteslot stornieren (AWS CLI)	19
Stornieren Sie einen Geräteslot (API)	19
Konzepte	20
Geräte	20
Unterstützte Geräte	20
Gerätepools	21
Private Geräte	21
Branding des Geräts	21
Steckplätze für Geräte	21
Vorinstallierte Geräte-Apps	22
Funktionen der Geräte	22
Testumgebungen	22

Standard-Testumgebung	22
Benutzerdefinierte Testumgebung	23
Ausführungen	23
Konfiguration ausführen	24
Führen Sie die Aufbewahrung von Dateien aus	24
Status des Geräts ausführen	24
Parallele Läufe	24
Einstellung des Ausführungs-Timeouts	24
Werbung in Läufen	25
Medien in Runs	25
Allgemeine Aufgaben für Läufe	25
Apps	25
Instrumentierung von Apps	25
Apps in Läufen erneut signieren	25
Verschleierte Apps in Läufen	26
Berichte	26
Aufbewahrung von Berichten	26
Komponenten des Berichts	26
Loggt in Berichten ein	27
Allgemeine Aufgaben für Berichte	27
Sitzungen	27
Unterstützte Geräte für den Fernzugriff	27
Aufbewahrung von Sitzungsdateien	27
Instrumentierung von Apps	28
Apps in Sitzungen erneut signieren	28
Verschleierte Apps in Sitzungen	28
Mit Projekten arbeiten	29
Erstellen eines Projekts	29
Voraussetzungen	29
Erstellen Sie ein Projekt (Konsole)	29
Erstellen Sie ein Projekt (AWS CLI)	30
Erstellen Sie ein Projekt (API)	30
Sehen Sie sich die Projektliste an	30
Voraussetzungen	31
Sehen Sie sich die Projektliste an (Konsole)	31
Sehen Sie sich die Projektliste an (AWS CLI)	31

Sehen Sie sich die Projektliste an (API)	31
Arbeiten mit Testläufen	32
Erstellen Sie einen Testlauf	32
Voraussetzungen	33
Erstellen Sie einen Testlauf (Konsole)	33
Erstellen Sie einen Testlauf (AWS CLI)	36
Erstellen Sie einen Testlauf (API)	46
Nächste Schritte	47
Legen Sie das Ausführungs-Timeout fest	47
Voraussetzungen	48
Legen Sie das Ausführungs-Timeout für ein Projekt fest	48
Legen Sie das Ausführungs-Timeout für einen Testlauf fest	49
Simulieren Sie Netzwerkverbindungen und -bedingungen	49
Richten Sie Network Shaping ein, wenn Sie einen Testlauf planen	50
Erstellen Sie ein Netzwerkprofil	51
Ändern Sie die Netzwerkbedingungen während des Tests	52
Stoppen Sie einen Lauf	52
Stoppen Sie einen Lauf (Konsole)	52
Stoppen Sie einen Lauf (AWS CLI)	54
Stoppen Sie einen Lauf (API)	56
Eine Liste der Läufe anzeigen	56
Eine Liste der Läufe anzeigen (Konsole)	56
Eine Liste von Läufen anzeigen (AWS CLI)	57
Eine Liste der Läufe anzeigen (API)	57
Erstellen eines Gerätepools	57
Voraussetzungen	57
Erstellen Sie einen Gerätepool (Konsole)	58
Erstellen Sie einen Gerätepool (AWS CLI)	59
Erstellen Sie einen Gerätepool (API)	59
Analysieren der Ergebnisse	59
Mit Testberichten arbeiten	60
Mit Artefakten arbeiten	70
Taggen in der Gerätefarm	75
Markieren von Ressourcen	75
Ressourcen nach Tag nachschlagen	76
Entfernen von Tags von Ressourcen	77

Testtypen und Frameworks	78
Frameworks testen	78
Frameworks zum Testen von Android-Anwendungen	78
Frameworks zum Testen von iOS-Anwendungen	78
Frameworks zum Testen von Webanwendungen	78
Frameworks in einer benutzerdefinierten Testumgebung	78
Unterstützung der Appium-Version	78
Integrierte Testtypen	79
Appium	79
Versionsunterstützung	79
Konfigurieren Sie Ihr Appium-Testpaket	80
Erstellen Sie eine komprimierte Paketdatei	91
Laden Sie Ihr Testpaket auf Device Farm hoch	94
Machen Sie Screenshots Ihrer Tests (optional)	95
Android-Tests	96
Frameworks zum Testen von Android-Anwendungen	96
Integrierte Testtypen für Android	96
Instrumentierung	96
iOS-Tests	99
Frameworks zum Testen von iOS-Anwendungen	99
Integrierte Testtypen für iOS	100
XCTest	100
XCTest-UI	102
Tests für Webanwendungen	104
Regeln für Geräte mit und ohne Messgerät	104
Integrierte Tests	105
Integrierte Testtypen	105
Eingebaut: Fuzz (Android und iOS)	105
Arbeiten mit benutzerdefinierten Testumgebungen	108
Syntax der Testspezifikation	109
Beispiel für eine Testspezifikation	111
Android-Testumgebung	117
Unterstützte Software	118
devicefarm-cli	119
Auswahl des Android-Testhosts	120
Beispiel für eine Testspezifikationsdatei	122

Migration zu Amazon Linux 2 Test Host	126
Umgebungsvariablen	129
Allgemeine Umgebungsvariablen	129
Appium-Java-JUnit-Umgebungsvariablen	131
Appium-Java-TestNG-Umgebungsvariablen	131
XCUI-Test-Umgebungsvariablen	132
Migrieren von Tests	132
Überlegungen bei der Migration	132
Schritte zur Migration	134
Appium-Framework	135
Android-Instrumentierung	135
Migrieren vorhandener iOS XCUI-Test-Tests	135
Erweiterung des benutzerdefinierten Modus	135
Eine PIN festlegen	135
Beschleunigung von APPIUM-basierten Tests durch die gewünschten Funktionen	136
Verwenden Sie Webhooks und andere APIs nach der Ausführung Ihrer Tests	139
Zusätzliche Dateien zu Ihrem Testpaket hinzufügen	140
Arbeiten mit Fernzugriff	144
Erstellen Sie eine Sitzung	144
Voraussetzungen	145
Erstellen Sie eine Sitzung mit der Device Farm-Konsole	145
Nächste Schritte	146
Verwenden Sie eine Sitzung	146
Voraussetzungen	146
Verwenden Sie eine Sitzung in der Device Farm-Konsole	146
Nächste Schritte	147
Tipps und Tricks	147
Sitzungsergebnisse abrufen	148
Voraussetzungen	148
Sitzungsdetails anzeigen	148
Sitzungsvideos oder Sitzungsprotokolle werden heruntergeladen	148
Arbeiten mit privaten Geräten	149
Verwaltung privater Geräte	150
Erstellen eines Instance-Profiles	150
Eine private Geräteinstanz verwalten	152
Erstellen Sie einen Testlauf oder eine Fernzugriffssitzung	154

Nächste Schritte	155
Private Geräte auswählen	155
ARN-Regeln für Geräte	156
Regeln für Labels von Geräteinstanzen	157
ARN-Regeln für Instanzen	157
Erstellen Sie einen privaten Gerätepool	158
Einen privaten Gerätepool mit privaten Geräten erstellen (AWS CLI)	160
Erstellen eines privaten Gerätepools mit privaten Geräten (API)	161
Das erneute Signieren von Apps wird übersprungen	161
Überspringen Sie das erneute Signieren von Apps auf Android-Geräten	164
Überspringen Sie das erneute Signieren von Apps auf iOS-Geräten	164
Erstellen Sie eine Fernzugriffssitzung, um Ihrer App zu vertrauen	165
Regionsübergreifendes Arbeiten	166
Überblick über das VPC-Peering	167
Voraussetzungen	168
Schritt 1: Stellen Sie eine Peering-Verbindung zwischen zwei VPCs her	169
Schritt 2: Aktualisieren Sie die Routentabellen für VPC-1 und VPC-2	169
Schritt 3: Zielgruppen erstellen	170
Schritt 4: Erstellen Sie einen Network Load Balancer	172
Schritt 5: Erstellen Sie einen VPC-Endpunktdienst	173
Schritt 6: Erstellen Sie eine VPC-Endpunktconfiguration in der Anwendung	173
Schritt 7: Erstellen Sie einen Testlauf	174
Schaffung skalierbarer VPC-Systeme	174
Kündigung privater Geräte	174
VPC-Konnektivität	175
AWS Zugriffskontrolle und IAM	177
Service-verknüpfte Rollen	178
Dienstbezogene Rollenberechtigungen für Device Farm	179
Eine dienstverknüpfte Rolle für Device Farm erstellen	182
Bearbeiten einer dienstverknüpften Rolle für Device Farm	182
Löschen einer dienstverknüpften Rolle für Device Farm	182
Unterstützte Regionen für mit Device Farm Farm-Dienst verknüpfte Rollen	183
Voraussetzungen	184
Verbindung zu Amazon VPC herstellen	185
Einschränkungen	187
Verwenden von VPC-Endpunktdiensten — Legacy	187

Bevor Sie beginnen	188
Schritt 1: Einen Network Load Balancer erstellen	189
Schritt 2: Erstellen Sie einen VPC-Endpunktdienst	192
Schritt 3: VPC-Endpunktconfiguration erstellen	193
Schritt 4: Erstellen Sie einen Testlauf	194
Protokollierung von API-Aufrufen mit AWS CloudTrail	195
Informationen zur AWS-Gerätefarm inCloudTrail	195
Grundlegendes zu den Einträgen der AWS Device Farm-Protokolldateien	196
CodePipeline-Integration	199
konfigurierenCodePipelineum Ihre Device Farm-Tests zu verwenden	200
AWS CLI-Referenz	205
WindowsPowerShellreferenzieren	206
Gerätefarm wird automatisiert	207
Beispiel: Verwendung desAWSSDK zum Starten einer Gerätefarm, zum Ausführen und Sammeln von Artefakten	207
Fehlerbehebung	212
Android-Anwendungen	212
ANDROID_APP_UNZIP_FAILED	212
ANDROID_APP_AAPT_DEBUG_BADGING_FAILED	213
ANDROID_APP_PACKAGE_NAME_VALUE_MISSING	215
ANDROID_APP_SDK_VERSION_VALUE_MISSING	215
ANDROID_APP_AAPT_DUMP_XMLTREE_FAILED	216
ANDROID_APP_DEVICE_ADMIN_PERMISSIONS	217
Bestimmte Fenster in meiner Android-Anwendung zeigen einen leeren oder schwarzen Bildschirm	219
Appium Java JUnit	220
APPIUM_JAVA_JUNIT_TEST_PACKAGE_PACKAGE_UNZIP_FAILED	220
APPIUM_JAVA_JUNIT_TEST_PACKAGE_DEPENDENCY_DIR_MISSING	221
APPIUM_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DIR	222
APPIUM_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING	223
APPIUM_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JAR	224
APPIUM_JAVA_JUNIT_TEST_PACKAGE_JUNIT_VERSION_VALUE_UNKNOWN	226
APPIUM_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION	227
Appium Java Unit-Webseite	229
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED	229
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_DEPENDENCY_DIR_MISSING	230

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DIR ..	231
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING	232
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JAR	233
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_JUNIT_VERSION_VALUE_UNKNOWN ...	235
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION	236
Appium Java TestNG	237
APPIUM_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED	237
APPIUM_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSING	238
APPIUM_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DIR	240
APPIUM_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING	241
APPIUM_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JAR	242
Appium, Java, TestNG, Webseite	244
APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED	244
APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSING	245
APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DIR	246
APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING	247
APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JAR	248
Appium Python	250
APPIUM_PYTHON_TEST_PACKAGE_UNZIP_FAILED	250
APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING	251
APPIUM_PYTHON_TEST_PACKAGE_INVALID_PLATFORM	252
APPIUM_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING	253
APPIUM_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME	254
APPIUM_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISSING	255
APPIUM_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION	257
APPIUM_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS_FAILED	258
APPIUM_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED	259
APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEELS_INSUFFICIENT	261
Appium Python-Web	262
APPIUM_WEB_PYTHON_TEST_PACKAGE_UNZIP_FAILED	262
APPIUM_WEB_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING	263
APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PLATFORM	264
APPIUM_WEB_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING	265
APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME	267
APPIUM_WEB_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISSING	268
APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION	269

APPIUM_WEB_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS_FAILED	270
APPIUM_WEB_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED	272
Instrumentierung	273
INSTRUMENTATION_TEST_PACKAGE_UNZIP_FAILED	273
INSTRUMENTATION_TEST_PACKAGE_AAPT_DEBUG_BADGING_FAILED	274
INSTRUMENTATION_TEST_PACKAGE_INSTRUMENTATION_RUNNER_VALUE_MISSING	275
INSTRUMENTATION_TEST_PACKAGE_AAPT_DUMP_XMLTREE_FAILED	277
INSTRUMENTATION_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSING	278
iOS-Anwendungen	279
IOS_APP_UNZIP_FAILED	279
IOS_APP_PAYLOAD_DIR_MISSING	280
IOS_APP_APP_DIR_MISSING	281
IOS_APP_PLIST_FILE_MISSING	282
IOS_APP_CPU_ARCHITECTURE_VALUE_MISSING	283
IOS_APP_PLATFORM_VALUE_MISSING	284
IOS_APP_WRONG_PLATFORM_DEVICE_VALUE	286
IOS_APP_FORM_FACTOR_VALUE_MISSING	287
IOS_APP_PACKAGE_NAME_VALUE_MISSING	288
IOS_APP_EXECUTABLE_VALUE_MISSING	290
XCTest	291
XCTEST_TEST_PACKAGE_UNZIP_FAILED	292
XCTEST_TEST_PACKAGE_XCTEST_DIR_MISSING	292
XCTEST_TEST_PACKAGE_PLIST_FILE_MISSING	293
XCTEST_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING	294
XCTEST_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING	296
XCTest-UI	297
XCTEST_UI_TEST_PACKAGE_UNZIP_FAILED	297
XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_MISSING	298
XCTEST_UI_TEST_PACKAGE_APP_DIR_MISSING	299
XCTEST_UI_TEST_PACKAGE_PLUGINS_DIR_MISSING	300
XCTEST_UI_TEST_PACKAGE_XCTEST_DIR_MISSING_IN_PLUGINS_DIR	301
XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING	302
XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING_IN_XCTEST_DIR	303
XCTEST_UI_TEST_PACKAGE_CPU_ARCHITECTURE_VALUE_MISSING	304
XCTEST_UI_TEST_PACKAGE_PLATFORM_VALUE_MISSING	306
XCTEST_UI_TEST_PACKAGE_WRONG_PLATFORM_DEVICE_VALUE	307

XCTEST_UI_TEST_PACKAGE_FORM_FACTOR_VALUE_MISSING	309
XCTEST_UI_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING	310
XCTEST_UI_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING	312
XCTEST_UI_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSING	313
XCTEST_UI_TEST_PACKAGE_TEST_EXECUTABLE_VALUE_MISSING	315
Sicherheit	317
Identity and Access Management	318
Zielgruppe	318
Authentifizierung mit Identitäten	318
Funktionsweise von AWS Device Farm mit IAM	322
Verwalten des Zugriffs mit Richtlinien	327
Beispiele für identitätsbasierte Richtlinien	330
Fehlerbehebung	335
Compliance-Validierung	338
Datenschutz	339
Verschlüsselung während der Übertragung	340
Verschlüsselung im Ruhezustand	340
Datenaufbewahrung	340
Datenverwaltung	341
Schlüsselverwaltung	342
Richtlinie für den Datenverkehr zwischen Netzwerken	342
Ausfallsicherheit	342
Sicherheit der Infrastruktur	343
Sicherheit der Infrastruktur für Tests physischer Geräte	344
Infrastruktursicherheit für das Testen von Desktop-Browsern	344
Konfigurations- und Schwachstellenanalyse	344
Vorfallreaktion	345
Protokollierung und Überwachung	346
Bewährte Methoden für die Gewährleistung der Sicherheit	346
Einschränkungen	347
Tools und Plugins	348
Jenkins CI-Plugin	348
Schritt 1: Installieren des Plugins	351
Schritt 2: Erstellen eines IAM-Benutzers	352
Schritt 3: Anleitung zur initialen Konfigurierung	354
Schritt 4: Verwenden Sie das Plugin	354

Abhängigkeiten	355
Gradle-Plugin für Gerätefarm	355
Erstellen des Device Farm Gradle-Plug-ins	356
Einrichtung des Device Farm Gradle-Plug-ins	356
Generieren eines IAM-Benutzers	359
Konfigurieren von Testtypen	360
Abhängigkeiten	362
Dokumentverlauf	363
AWS-Glossar	368
.....	ccclxix

Was ist AWS Device Farm?

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

Device Farm kann hauptsächlich auf zwei Arten verwendet werden:

- Automatisierte Tests von Apps mit verschiedenen verfügbaren Test-Frameworks.
- Remote-Zugriff auf Geräte, auf denen Sie in Echtzeit Apps laden, ausführen und mit diesen interagieren können.

Note

Device Farm ist nur verfügbar in us-west-2 Region (Oregon).

Automatisiertes Testen von Apps

Device Farm ermöglicht es Ihnen, Ihre eigenen Tests hochzuladen oder integrierte, skriptfreie Kompatibilitätstests zu verwenden. Da Tests parallel durchgeführt werden, starten Tests auf mehreren Geräten innerhalb von wenigen Minuten.

Wenn die Tests abgeschlossen sind, ein Testbericht mit Ergebnissen auf hoher Ebene und Protokollen auf niedriger Ebene, pixel-to-pixel Screenshots und Leistungsdaten werden aktualisiert.

Device Farm unterstützt das Testen von nativen und hybriden Android- und iOS-Apps, einschließlich solcher, die mit PhoneGap, Titanium, Xamarin, Unity und andere Frameworks. Der Remote-Zugriff auf Android- und iOS-Apps für interaktive Tests wird nicht unterstützt. Weitere Informationen zu unterstützten Testtypen finden Sie unter [Arbeiten mit Testtypen in AWS Device Farm](#).

Interaktion mit Fernzugriff

Mit dem Remotezugriff können Sie in Echtzeit über Ihren Webbrowser durch Wischen und Gesten mit einem Gerät interagieren. Es gibt eine Reihe von Situationen, in denen Echtzeit-Interaktion mit einem Gerät nützlich sind. Beispielsweise kann ein Mitarbeiter im Kundenservice Kunden

durch die Verwendung oder Einrichtung eines Geräts leiten. Darüber hinaus kann er den Kunden in Bezug auf die Nutzung von Apps auf einem spezifischen Gerät anleiten. Sie können Apps auf einem Gerät installieren, welches in einer Remotezugriffssitzung ausgeführt wird, und anschließend Kundenprobleme oder gemeldete Fehler reproduzieren.

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

Terminologie

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

Gerätepools

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

Auftrag

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

Gebührenerfassung

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

project

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

Bericht

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

run

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

Sitzung

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

Suite

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

Test

Ein einzelner Testfall in einem Testpaket.

Weitere Informationen über eine Device Farm finden Sie unter [Konzepte](#).

Einrichtung

Informationen zur Verwendung von Device Farm finden Sie unter [Einrichtung](#).

Einrichtung von AWS Device Farm

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

Themen

- [Schritt 1: Melden Sie sich an für AWS](#)
- [Schritt 2: Erstellen oder verwenden Sie einen IAM-Benutzer in Ihrem Konto AWS](#)
- [Schritt 3: Erteilen Sie dem IAM-Benutzer die Erlaubnis, auf Device Farm zuzugreifen](#)
- [Nächster Schritt](#)

Schritt 1: Melden Sie sich an für AWS

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

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

Um sich für eine anzumelden AWS-Konto

1. Öffnen Sie <https://portal.aws.amazon.com/billing/signup>.
2. Folgen Sie den Online-Anweisungen.

Bei der Anmeldung müssen Sie auch einen Telefonanruf entgegennehmen und einen Verifizierungscode über die Telefontasten eingeben.

Wenn Sie sich für eine anmelden AWS-Konto, Root-Benutzer des AWS-Kontos wird eine erstellt. Der Root-Benutzer hat Zugriff auf alle AWS-Services und Ressourcen des Kontos. Aus Sicherheitsgründen sollten Sie einem Benutzer Administratorzugriff zuweisen und nur den Root-Benutzer verwenden, um [Aufgaben auszuführen, für die Root-Benutzerzugriff erforderlich](#) ist.

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

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

Weitere Informationen finden Sie unter [Einen IAM-Benutzer erstellen](#) ().AWS Management Console

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

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

Note

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

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

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

Weitere Informationen zum Erstellen von IAM-Richtlinien finden Sie unter [Erstellen von IAM-Richtlinien](#) im IAM-Benutzerhandbuch.

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

Durch das Anhängen der Richtlinie erhält der IAM-Benutzer Zugriff auf alle Device Farm Farm-Aktionen und Ressourcen, die diesem IAM-Benutzer zugeordnet sind. Informationen dazu, wie Sie IAM-Benutzer auf eine begrenzte Anzahl von Gerätefarmaktionen und -ressourcen beschränken können, finden Sie unter [Identity and Access Management in AWS Device Farm](#).

Nächster Schritt

Sie sind jetzt bereit, Device Farm zu verwenden. Siehe [Erste Schritte mit Device Farm](#).

Erste Schritte mit Device Farm

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

Note

Device Farm ist nur verfügbar in us-west-2 (Oregon) AWS Region.

Themen

- [Voraussetzungen](#)
- [Schritt 1: Melden Sie sich bei der -Konsole an](#)
- [Schritt 2: Erstellen Sie ein Projekt](#)
- [Schritt 3: Einen Lauf erstellen und starten](#)
- [Schritt 4: Sehen Sie sich die Ergebnisse des Testlaufs an](#)
- [Nächste Schritte](#)

Voraussetzungen

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

- Führen Sie die Schritte unter [Einrichtung](#) aus. Du brauchst ein AWS-Konto und ein AWS Identity and Access Management (IAM-) Benutzer mit der Berechtigung, auf Device Farm zuzugreifen.
- Für Android benötigen Sie eine .apk (Android App Package)-Datei. Für iOS benötigen Sie eine .ipa (iOS App Archive)-Datei. Sie laden die Datei später in dieser exemplarischen Vorgehensweise auf Device Farm hoch.

Note

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

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

Schritt 1: Melden Sie sich bei der -Konsole an

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

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

Schritt 2: Erstellen Sie ein Projekt

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

1. Wählen Sie im Navigationsbereich Testen von Mobilgeräten, und wählen Sie dann Projekte.
2. Unter Projekte zum Testen von Mobilgeräten, wählen Neues Projekt.
3. Unter Projekt erstellen, geben Sie ein Name des Projekts (zum Beispiel **MyDemoProject**).
4. Wählen Sie Erstellen aus.

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

Schritt 3: Einen Lauf erstellen und starten

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

1. Wählen Sie auf der Seite Automated tests (Automatisierte Tests) die Option Create a new run (Einen neuen Lauf erstellen).
2. Auf dem Wählen Sie die Anwendung Seite, unter App für Mobilgeräte, wählen Wählen Sie Datei und wählen Sie dann eine Android- (.apk) oder iOS- (.ipa) -Datei von Ihrem Computer aus. Oder ziehen Sie die Datei von Ihrem Computer und legen Sie sie in der Konsole ab.

3. Geben Sie ein Namen ausführen, wiemy **first test**. Standardmäßig verwendet die Device Farm-Konsole den Dateinamen.
 4. Wählen Sie Weiter aus.
 5. Auf dem Konfigurieren Seite, unter Test-Framework einrichten, wählen Sie eines der Test-Frameworks oder der integrierten Testsuiten. Informationen zu den jeweiligen Optionen finden Sie unter [Testtypen und Frameworks](#).
 - Wenn Sie Ihre Tests noch nicht für Device Farm gepackt haben, wählen Sie Eingebaut: Fuzzum eine standardmäßige, integrierte Testsuite auszuführen. Sie können die Standardwerte für beibehalten Anzahl der Ereignisse, Drosselung von Ereignissen, und Randomizer-Seed. Weitere Informationen finden Sie unter [the section called “Eingebaut: Fuzz \(Android und iOS\)”](#).
 - Wenn Sie ein Testpaket aus einem der unterstützten Test-Frameworks haben, wählen Sie das entsprechende Test-Framework aus und laden Sie dann die Datei hoch, die Ihre Tests enthält.
 6. Wählen Sie Weiter aus.
 7. Auf dem Wählen Sie Geräte aus Seite, für Gerätepool, wählen Die besten Geräte.
 8. Wählen Sie Weiter aus.
 9. Führen Sie auf der Seite Specify device state (Gerätestatus angeben) eine der folgenden Aktionen durch:
 - Um zusätzliche Daten bereitzustellen, die Device Farm während der Ausführung verwenden kann, finden Sie unter Zusätzliche Daten hinzufügen, laden Sie eine ZIP-Datei hoch.
 - Um andere Apps für den Lauf zu installieren, finden Sie unter Installieren Sie andere Apps, laden Sie die APK- oder IPA-Dateien für die Apps hoch. Um die Installationsreihenfolge zu ändern, ziehen Sie die Dateien per Drag & Drop.
 - Um WLAN-, Bluetooth-, GPS- oder NFC-Funkgeräte für den Lauf einzuschalten, finden Sie unter Stellen Sie die Funkstatus ein, wählen Sie die entsprechenden Kontrollkästchen aus.
-  **Note**

Das Einstellen des Funkstatus des Geräts ist derzeit nur für native Android-Tests verfügbar.
- Um das standortspezifische Verhalten während der Ausführung zu testen, finden Sie unter Standort des Geräts, Voreinstellung angeben Breitengrad und Längengrad Koordinaten.

- Um Gerätesprache und Region für den Lauf voreinzustellen, klicken Sie unterGebietsschema des Geräts, wählen Sie ein Gebietsschema.
- Um das Netzwerkprofil für den Lauf voreinzustellen, klicken Sie unterNetzwerkprofil, wählen Sie ein kuratiertes Profil. Oder wähleNetzwerkprofil erstellenum Ihr eigenes zu erstellen.

10. Wählen Sie Weiter aus.

11. Wählen Sie auf der Seite Review and start run (Testlauf überprüfen und starten) die Option Confirm and start run (Testlauf bestätigen und starten).

Device Farm startet die Ausführung, sobald Geräte verfügbar sind, normalerweise innerhalb weniger Minuten. Um den Ausführungsstatus anzuzeigen, klicken Sie aufAutomatisierte TestsWählen Sie auf der Seite Ihres Projekts den Namen Ihres Laufs aus. Auf der Run-Seite, unterGeräte, jedes Gerät beginnt mit dem ausstehenden

Symbol 

der Gerätetabelle und wechselt dann zum

Laufsymbol 

der Test beginnt. Nach Abschluss jedes Tests zeigt die Konsole neben dem Gerätenamen ein Testergebnissymbol an. Wenn alle Tests abgeschlossen sind, ändert sich das Symbol für ausstehende Tests neben dem Testlauf in ein Testergebnissymbol.

Schritt 4: Sehen Sie sich die Ergebnisse des Testlaufs an

Um die Testergebnisse des Laufs anzuzeigen, klicken Sie aufAutomatisierte TestsWählen Sie auf der Seite Ihres Projekts den Namen Ihres Laufs aus. Eine Übersichtsseite wird angezeigt:

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

Weitere Informationen finden Sie unter [Arbeiten mit Testberichten in Device Farm](#).

Nächste Schritte

Weitere Informationen über eine Device Farm finden Sie unter [Konzepte](#).

Erwerben Sie einen Geräteslot in Device Farm

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

Themen

- [Geräteslots \(Konsole\) kaufen](#)
- [Erwerben Sie einen Geräteslot \(AWS CLI\)](#)
- [Erwerben Sie einen Geräteslot \(API\)](#)
- [Stornieren Sie einen Geräteslot \(Konsole\)](#)
- [Einen Geräteslot stornieren \(AWS CLI\)](#)
- [Stornieren Sie einen Geräteslot \(API\)](#)

Geräteslots (Konsole) kaufen

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

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

Important

Wenn Sie die Anzahl der Geräteslots ändern, aber die Nachricht „Kontaktieren Sie uns“ oder „Kontaktieren Sie uns für einen Kauf“ erhalten, ist Ihr AWS Konto noch nicht für den Kauf der von Ihnen angeforderten Anzahl von Geräteslots zugelassen.

Bei diesen Optionen werden Sie aufgefordert, eine E-Mail an das Device Farm Farm-Supportteam zu senden. Geben Sie in der E-Mail die Nummer jedes Gerätetyps an, den Sie kaufen möchten, und für welchen Abrechnungszeitraum.

Note

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

Purchase and manage device slots

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

Automated testing

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

Current billing period

You currently have

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

Next billing period

From August 16, you will have

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

Remote access

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

Current billing period

You currently have

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

Next billing period

From August 16, you will have

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

Save

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

Confirm purchase



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

Cancel

Confirm

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

Erwerben Sie einen Geräteslot (AWS CLI)

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

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

```
{
  "accountSettings": {
    "maxSlots": {
      "GUID": 1,
      "GUID": 1,
      "GUID": 1,
      "GUID": 1
    },
    "unmeteredRemoteAccessDevices": {
```

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

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

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

```

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

```

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

Note

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

Die Ausgabe sollte folgendermaßen oder ähnlich aussehen:

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

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

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

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

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

Erwerben Sie einen Geräteslot (API)

1. Rufen Sie den Vorgang [GetAccountEinstellungen](#) auf, um Ihre Kontoeinstellungen aufzulisten.
2. Rufen Sie den [ListOfferings](#) Betrieb auf, um eine Liste der verfügbaren Steckplätze für Geräte aufzulisten.
3. Rufen Sie den [ListOfferingBereich Werbeaktionen](#) auf, um eine Liste der verfügbaren Angebote zu erhalten.

Note

Mit diesem Befehl werden nur Werbeaktionen zurückgegeben, die Sie noch nicht gekauft haben. Sobald Sie einen oder mehrere Plätze über eine Werbeaktion erwerben, erscheint diese Aktion nicht mehr in den Suchergebnissen.

4. Rufen Sie den [PurchaseOffering](#) Betrieb auf, um ein Angebot zu erwerben.
5. Rufen Sie den Vorgang [GetOfferingStatus](#) auf, um den Angebotsstatus abzurufen.

Die Befehle [RenewOffering](#) und [ListOfferingTransactions](#) sind auch für diese Funktion verfügbar.

Hinweise zur Verwendung der Device Farm API finden Sie unter [Gerätefarm wird automatisiert](#).

Stornieren Sie einen Geräteslot (Konsole)

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

Einen Geräteslot stornieren (AWS CLI)

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

Stornieren Sie einen Geräteslot (API)

Rufen Sie den [RenewOffering](#) Vorgang auf, um die Anzahl der Geräte in Ihrem Konto zu ändern.

Konzepte von AWS Device Farm

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

- [Geräteunterstützung in AWS Device Farm](#)
- [Testumgebungen](#)
- [Ausführungen](#)
- [Apps](#)
- [Berichte in AWS Device Farm](#)
- [Sitzungen](#)

Weitere Informationen zu den unterstützten Testtypen in Device Farm finden Sie unter [Arbeiten mit Testtypen in AWS Device Farm](#).

Geräteunterstützung in AWS Device Farm

Die folgenden Abschnitte enthalten Informationen zur Geräteunterstützung in Device Farm.

Themen

- [Unterstützte Geräte](#)
- [Gerätepools](#)
- [Private Geräte](#)
- [Branding des Geräts](#)
- [Steckplätze für Geräte](#)
- [Vorinstallierte Geräte-Apps](#)
- [Funktionen der Geräte](#)

Unterstützte Geräte

Device Farm bietet Unterstützung für Hunderte einzigartiger, beliebter Android- und iOS-Geräte und Betriebssystemkombinationen. Die Liste der verfügbaren Geräte wird immer größer, wenn neue Geräte auf den Markt kommen. Die vollständige Liste der Geräte finden unter [Geräteliste](#).

Gerätepools

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

Private Geräte

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

Branding des Geräts

Device Farm führt Tests auf physischen Mobil- und Tablet-Geräten verschiedener OEMs durch.

Steckplätze für Geräte

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

Es gibt zwei Typen von Geräteplätzen:

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

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

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

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

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

Vorinstallierte Geräte-Apps

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

Funktionen der Geräte

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

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

Google Play Services ist auf den Geräten installiert, die den Service unterstützen, aber diese Geräten verfügen über kein aktives Google-Konto.

Testumgebungen in AWS Device Farm

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

Themen

- [Standard-Testumgebung](#)
- [Benutzerdefinierte Testumgebung](#)

Standard-Testumgebung

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

Note

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

Benutzerdefinierte Testumgebung

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

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

Weitere Informationen finden Sie unter [Hochladen einer benutzerdefinierten Testspezifikation mithilfe von](#) und [AWS CLI Erstellen Sie einen Testlauf in Device Farm](#)

Läuft in der AWS-Gerätefarm

Die folgenden Abschnitte enthalten Informationen zu Ausführungen in Device Farm.

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

Themen

- [Konfiguration ausführen](#)
- [Führen Sie die Aufbewahrung von Dateien aus](#)
- [Status des Geräts ausführen](#)

- [Parallele Läufe](#)
- [Einstellung des Ausführungs-Timeouts](#)
- [Werbung in Läufen](#)
- [Medien in Runs](#)
- [Allgemeine Aufgaben für Läufe](#)

Konfiguration ausführen

Im Rahmen eines Laufs können Sie Einstellungen angeben, die Device Farm verwenden kann, um aktuelle Geräteeinstellungen zu überschreiben. Diese umfassen die Koordinaten für Längen- und Breitengrad, Gebietsschema, Radiostatus (z. B. Bluetooth, GPS, NFC und WLAN), zusätzliche Daten (in einer ZIP-Datei) und unterstützende Apps (Apps, die vor der zu testenden App installiert werden sollten).

Führen Sie die Aufbewahrung von Dateien aus

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

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

Status des Geräts ausführen

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

Parallele Läufe

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

Einstellung des Ausführungs-Timeouts

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

Weitere Informationen finden Sie unter [Legen Sie das Ausführungstimeout für Testläufe in AWS Device Farm fest](#).

Werbung in Läufen

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

Medien in Runs

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

Allgemeine Aufgaben für Läufe

Weitere Informationen finden Sie unter [Erstellen Sie einen Testlauf in Device Farm](#) und [Arbeiten mit Testläufen in AWS Device Farm](#).

Apps in der AWS-Gerätefarm

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

Themen

- [Instrumentierung von Apps](#)
- [Apps in Läufen erneut signieren](#)
- [Verschleierte Apps in Läufen](#)

Instrumentierung von Apps

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

Apps in Läufen erneut signieren

Für iOS-Apps müssen Sie Ihrem Provisioning-Profil keine Device Farm Farm-UUIDs hinzufügen. Device Farm ersetzt das eingebettete Bereitstellungsprofil durch ein Platzhalterprofil und signiert die App anschließend erneut. Wenn Sie Zusatzdaten bereitstellen, fügt Device Farm sie dem Paket der App hinzu, bevor Device Farm sie installiert, sodass die Hilfsdaten in der Sandbox Ihrer App vorhanden sind. Durch erneutes Signieren der App werden Berechtigungen wie App Group,

Associated Domains, Game Center,, HealthKit, Konfiguration von drahtlosem Zubehör HomeKit, In-App-Kauf, Inter-App-Audio, Apple Pay, Push-Benachrichtigungen und VPN-Konfiguration und -Steuerung entfernt.

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

Verschleierte Apps in Läufen

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

Berichte in AWS Device Farm

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

Themen

- [Aufbewahrung von Berichten](#)
- [Komponenten des Berichts](#)
- [Loggt in Berichten ein](#)
- [Allgemeine Aufgaben für Berichte](#)

Aufbewahrung von Berichten

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

Komponenten des Berichts

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

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

Loggt in Berichten ein

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

Allgemeine Aufgaben für Berichte

Weitere Informationen finden Sie unter [Arbeiten mit Testberichten in Device Farm](#).

Sitzungen in der AWS-Gerätefarm

Sie können Device Farm verwenden, um interaktive Tests von Android- und iOS-Apps über Fernzugriffssitzungen in einem Webbrowser durchzuführen. Diese Art von interaktiven Tests unterstützt Techniker bei Kundenanrufen dabei, das Problem des Kunden Schritt für Schritt durchzugehen. Entwickler können ein Problem auf einem spezifischen Gerät reproduzieren, um mögliche Quellen des Problems zu isolieren. Sie können Remotesitzungen verwenden, um mit Ihren Zielkunden Tests in Bezug auf die Benutzerfreundlichkeit auszuführen.

Themen

- [Unterstützte Geräte für den Fernzugriff](#)
- [Aufbewahrung von Sitzungsdateien](#)
- [Instrumentierung von Apps](#)
- [Apps in Sitzungen erneut signieren](#)
- [Verschleierte Apps in Sitzungen](#)

Unterstützte Geräte für den Fernzugriff

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

Aufbewahrung von Sitzungsdateien

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

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

Instrumentierung von Apps

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

Apps in Sitzungen erneut signieren

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

Verschleierte Apps in Sitzungen

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

Arbeiten mit Projekten in AWS Device Farm

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

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

Themen

- [Erstellen Sie ein Projekt in AWS Device Farm](#)
- [Sehen Sie sich die Projektliste in AWS Device Farm an](#)

Erstellen Sie ein Projekt in AWS Device Farm

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

Voraussetzungen

- Führen Sie die Schritte unter [Einrichtung](#) aus.

Erstellen Sie ein Projekt (Konsole)

1. Melden Sie sich bei der Device Farm-Konsole an unter <https://console.aws.amazon.com/devicefarm>.
2. Wählen Sie im Device Farm-Navigationsbereich Testen von Mobilgeräten und wählen Sie dann Projekte.
3. Wählen Sie New Project (Neues Projekt).
4. Geben Sie einen Namen für Ihr Projekt ein und wählen Sie Einreichen.
5. Um Einstellungen für das Projekt festzulegen, wählen Sie Project settings (Projekteinstellungen) aus. Diese Einstellungen umfassen die standardmäßige Zeitbeschränkung für Testläufe. Nachdem die Einstellungen angewendet wurden, werden sie von allen Testläufen für das Projekt

verwendet. Weitere Informationen finden Sie unter [Legen Sie das Ausführungszeitout für Testläufe in AWS Device Farm fest](#).

Erstellen Sie ein Projekt (AWS CLI)

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

Beispiel:

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

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

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

Weitere Informationen erhalten Sie unter [create-project](#) und [AWS CLI-Referenz](#).

Erstellen Sie ein Projekt (API)

- Rufen Sie die [CreateProject](#)-API auf.

Informationen zur Verwendung der Device Farm-API finden Sie unter [Gerätefarm wird automatisiert](#).

Sehen Sie sich die Projektliste in AWS Device Farm an

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

Themen

- [Voraussetzungen](#)

- [Sehen Sie sich die Projektliste an \(Konsole\)](#)
- [Sehen Sie sich die Projektliste an \(AWS CLI\)](#)
- [Sehen Sie sich die Projektliste an \(API\)](#)

Voraussetzungen

- Erstellen Sie mindestens ein Projekt in Device Farm. Befolgen Sie die Anweisungen unter [Erstellen Sie ein Projekt in AWS Device Farm](#), und kehren Sie dann zu dieser Seite zurück.

Sehen Sie sich die Projektliste an (Konsole)

1. Melden Sie sich bei der Device Farm-Konsole an unter <https://console.aws.amazon.com/devicefarm>.
2. Gehen Sie wie folgt vor, um die Liste der verfügbaren Projekte zu finden:
 - Wählen Sie für Testprojekte für mobile Geräte im Navigationsmenü Device Farm Testen von Mobilgeräten, dann wähle Projekte.
 - Für Projekte zum Testen von Desktop-Browsern wählen Sie im Device Farm-Navigationsmenü Testen von Desktop-Browsern, dann wähle Projekte.

Sehen Sie sich die Projektliste an (AWS CLI)

- Um die Projektliste anzuzeigen, führen Sie den Befehl [list-projects](#) aus.
Um Informationen zu einem einzigen Projekt anzuzeigen, führen Sie den Befehl [get-project](#) aus.

Informationen zur Verwendung von Device Farm mit dem AWS CLI, siehe [AWS CLI-Referenz](#).

Sehen Sie sich die Projektliste an (API)

- Um die Projektliste anzuzeigen, rufen Sie die [ListProjects](#)-API auf.
Um Informationen zu einem einzigen Projekt anzuzeigen, rufen Sie die [GetProject](#)-API auf.

Informationen zur AWS Device Farm-API finden Sie unter [Gerätefarm wird automatisiert](#).

Arbeiten mit Testläufen in AWS Device Farm

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

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

Themen

- [Erstellen Sie einen Testlauf in Device Farm](#)
- [Legen Sie das Ausführungstimeout für Testläufe in AWS Device Farm fest](#)
- [Simulieren Sie Netzwerkverbindungen und -bedingungen für Ihre AWS Device Farm Farm-Läufe](#)
- [Stoppen Sie einen Lauf in AWS Device Farm](#)
- [Eine Liste der Läufe in AWS Device Farm anzeigen](#)
- [Erstellen Sie einen Gerätepool in AWS Device Farm](#)
- [Analysieren der Ergebnisse in AWS Device Farm](#)

Erstellen Sie einen Testlauf in Device Farm

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

Themen

- [Voraussetzungen](#)
- [Erstellen Sie einen Testlauf \(Konsole\)](#)
- [Erstellen Sie einen Testlauf \(AWS CLI\)](#)
- [Erstellen Sie einen Testlauf \(API\)](#)
- [Nächste Schritte](#)

Voraussetzungen

Sie müssen ein Projekt in Device Farm haben. Befolgen Sie die Anweisungen unter [Erstellen Sie ein Projekt in AWS Device Farm](#), und kehren Sie dann zu dieser Seite zurück.

Erstellen Sie einen Testlauf (Konsole)

1. Melden Sie sich bei der Device Farm Farm-Konsole unter <https://console.aws.amazon.com/devicefarm> an.
2. Wählen Sie im Navigationsbereich Mobile Device Testing und dann Projects aus.
3. Wenn Sie bereits über ein Projekt verfügen, können Sie Ihre Tests in das Projekt hochladen. Wählen Sie andernfalls Neues Projekt aus, geben Sie einen Projektnamen ein und wählen Sie dann Erstellen aus.
4. Öffnen Sie Ihr Projekt und wählen Sie Create a new run (Neuen Lauf erstellen).
5. Wählen Sie auf der Seite „Anwendung auswählen“ die Option Mobile App oder Web-App aus.

Step 1
Choose application

Step 2
Configure

Step 3
Select devices

Step 4
Specify device state

Step 5
Review and start run

Choose application

Mobile App | Web App

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

Choose File or drop file here or Select a recent upload

Cancel Next step

6. Laden Sie Ihre Anwendungsdatei hoch. Sie können Ihre Datei auch per Drag & Drop hochladen oder einen aktuellen Upload auswählen. Wenn Sie eine iOS-App hochladen, müssen Sie darauf achten, iOS device (iOS-Gerät) auszuwählen, und keinen Simulator.
7. (Optional) Geben Sie unter Run name (Name des Laufs) einen Namen ein. Standardmäßig verwendet Device Farm den Namen der App-Datei.
8. Wählen Sie Weiter aus.
9. Wählen Sie auf der Seite Configure (Konfigurieren) eine der verfügbaren Testreihen aus.

Note

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

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

Weitere Informationen zu den verfügbaren Testreihen finden Sie unter [Arbeiten mit Testtypen in AWS Device Farm](#).

10. Wenn Sie „Integriert: Fuzz“ nicht ausgewählt haben, wählen Sie „Datei auswählen“ und suchen Sie dann nach der Datei, die Ihre Tests enthält, und wählen Sie sie aus.
11. Wählen Sie für Ihre Testumgebung die Option Test in unserer Standardumgebung ausführen oder Test in einer benutzerdefinierten Umgebung ausführen aus. Weitere Informationen finden Sie unter [Testumgebungen](#).
12. Wenn Sie die Standard-Testumgebung verwenden, fahren Sie mit Schritt 13 fort. Wenn Sie eine benutzerdefinierte Testumgebung mit der YAML-Datei der Standard-Testspezifikation verwenden, fahren Sie mit Schritt 13 fort.
 - a. Wenn Sie die Standard-Testspezifikation in einer benutzerdefinierten Testumgebung bearbeiten möchten, wählen Sie Edit (Bearbeiten), um die Standard-YAML-Spezifikation zu aktualisieren.
 - b. Wenn Sie die Testspezifikation geändert haben, wählen Sie Als neu speichern, um sie zu aktualisieren.
13. Wenn Sie die Videoaufzeichnungs- oder Leistungsdaten-Erfassungsoptionen konfigurieren möchten, wählen Sie Advanced Configuration (Erweiterte Konfiguration).
 - a. Wählen Sie Videoaufnahme aktivieren, um während des Tests Videos aufzunehmen.
 - b. Wählen Sie Erfassung von App-Leistungsdaten aktivieren aus, um Leistungsdaten vom Gerät zu erfassen.

 Note

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

14. Wählen Sie Weiter aus.
15. Führen Sie auf der Seite Select devices (Geräte auswählen) einen der folgenden Schritte aus:

- Um einen integrierten Gerätepool zu wählen, für den die Tests ausgeführt werden sollen, wählen Sie für Device pool (Gerätepool) die Option Top Devices (Top-Geräte).
- Um einen eigenen Gerätepool zu erstellen, für den die Tests ausgeführt werden sollen, befolgen Sie die Anweisungen in [Erstellen eines Gerätepools](#) und kehren dann zu dieser Seite zurück.
- Wenn Sie zuvor bereits einen eigenen Gerätepool erstellt haben, wählen Sie für Device pool (Gerätepool) diesen Gerätepool aus.

Weitere Informationen finden Sie unter [Geräteunterstützung in AWS Device Farm](#).

16. Wählen Sie Weiter.

17. Auf der Seite Specify device state (Gerätestatus angeben):

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

18. Wählen Sie Weiter aus.

19. Auf der Seite „Ausführung überprüfen und starten“ können Sie das Ausführungs-Timeout für Ihren Testlauf angeben. Wenn Sie unbegrenzte Testplätze verwenden, bestätigen Sie, dass Run on unmetered slots (Auf nicht zählerüberwachten Plätzen ausführen) aktiviert ist.

20. Geben Sie einen Wert ein oder ändern Sie das Ausführungs-Timeout mit dem Schieberegler. Weitere Informationen finden Sie unter [Legen Sie das Ausführungs-Timeout für Testläufe in AWS Device Farm fest](#).

21. Wählen Sie Confirm and start run (Bestätigen und Testlauf starten).

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



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



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

Wird

Informationen zum Beenden des Testlaufs finden Sie unter [Stoppen Sie einen Lauf in AWS Device Farm](#).

Erstellen Sie einen Testlauf (AWS CLI)

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

Themen

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

Schritt 1: Wählen Sie ein Projekt

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

1. Führen Sie den Befehl aus, um Ihre Device Farm Farm-Projekte aufzulisten `list-projects`. Wenn Sie über kein Projekt verfügen, finden Sie weitere Informationen unter [Erstellen Sie ein Projekt in AWS Device Farm](#).

Beispiel:

```
aws devicefarm list-projects
```

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

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

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

Schritt 2: Wählen Sie einen Gerätepool

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

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

Beispiel:

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

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

```
{
  "devicePools": [
    {
      "rules": [
        {
          "attribute": "ARN",
          "operator": "IN",

```

```

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

```

2. Wählen Sie einen Gerätepool aus und notieren Sie sich den zugehörigen ARN.

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

Schritt 3: Laden Sie Ihre Anwendungsdatei hoch

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

- Ihren Projekt-ARN.
- Name Ihrer App-Datei.
- Typ des Uploads.

Weitere Informationen finden Sie unter [create-upload](#).

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

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

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

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

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

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

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

Weitere Informationen finden Sie unter [Hochladen von Objekten mithilfe vorsignierter URLs](#) im Amazon Simple Storage Service-Benutzerhandbuch.

4. Um den Status Ihres App-Uploads zu überprüfen, führen Sie `get-upload` unter Angabe des ARN des App-Uploads aus.

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

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

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

Schritt 4: Laden Sie Ihr Testskript-Paket hoch

Als Nächstes laden Sie Ihr Testskript-Paket hoch.

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

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

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

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

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

```
}
```

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

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

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

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

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

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

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

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

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

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

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

Die Antwort enthält einen Eintrag für jede Standard-Testspezifikation:

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

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

Beispiel:

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

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

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

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

5. Sie können die Standard-Testspezifikation entsprechend Ihrer Testanforderungen bearbeiten und die abgewandelte Testspezifikation dann in zukünftigen Testläufen verwenden. Überspringen Sie diesen Schritt, um die Standard-Testspezifikation unverändert in einer benutzerdefinierten Testumgebung zu verwenden.
6. Um einen Upload Ihrer benutzerdefinierten Testspezifikation zu erstellen, führen Sie `create-upload` unter Angabe des Namens und Typs Ihrer Testspezifikation und des Projekt-ARN aus.

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

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

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

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

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

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

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

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

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

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

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

Schritt 6: Einen Testlauf planen

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

- Projekt-ARN von [Schritt 1](#).
- Gerätepool-ARN von [Schritt 2](#).
- App-Upload-ARN von [Schritt 3](#).
- Testpaket-Upload-ARN von [Schritt 4](#).

Wenn Sie Tests in einer benutzerdefinierten Umgebung ausführen, benötigen Sie außerdem Ihren Testspezifikation-ARN von [Schritt 5](#).

So planen Sie einen Testlauf in einer Standard-Testumgebung

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

Beispiel:

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

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

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

```

```
        "warned": 0,  
        "failed": 0,  
        "stopped": 0,  
        "passed": 0,  
        "errored": 0,  
        "total": 0  
    }  
}
```

Weitere Informationen finden Sie unter [schedule-run](#).

So planen Sie einen Testlauf in einer benutzerdefinierten Testumgebung

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

Beispiel:

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

So überprüfen Sie den Status Ihres Testlaufs

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

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

Weitere Informationen finden Sie unter [get-run](#). Informationen zur Verwendung von Device Farm mit dem AWS CLI finden Sie unter [AWS CLI-Referenz](#).

Erstellen Sie einen Testlauf (API)

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

Sie benötigen folgende Informationen zum Aufrufen der [ScheduleRun](#) API:

- Projekt-ARN. Siehe [Erstellen Sie ein Projekt \(API\)](#) und [CreateProject](#).
- App-Upload-ARN. Siehe [CreateUpload](#).
- Testpaket Upload-ARN. Siehe [CreateUpload](#).
- Gerätepool-ARN. Siehe [Erstellen eines Gerätepools](#) und [CreateDevicePool](#).

Note

Wenn Sie Tests in einer benutzerdefinierten Testumgebung ausführen, benötigen Sie außerdem Ihren Testspezifikation-Upload-ARN. Weitere Informationen finden Sie unter [Schritt 5: \(Optional\) Laden Sie Ihre benutzerdefinierte Testspezifikation hoch](#) und [CreateUpload](#).

Hinweise zur Verwendung der Device Farm API finden Sie unter [Gerätefarm wird automatisiert](#).

Nächste Schritte

In der Device Farm Farm-Konsole



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

Erfolg 

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

Befolgen Sie zur Nutzung des Berichts die Anweisungen unter [Arbeiten mit Testberichten in Device Farm](#).

Legen Sie das Ausführungstimeout für Testläufe in AWS Device Farm fest

Sie können über einen Wert festlegen, wie lange ein Testlauf ausgeführt werden soll, bevor Sie den Testlauf auf den Geräten stoppen. Der Standardwert des Ausführungstimeouts ist 150 Minuten pro Gerät, aber Sie können den Wert auf bis zu 5 Minuten heruntersetzen. Sie können die AWS

Device Farm Farm-Konsole oder die AWS Device Farm Farm-API verwenden AWS CLI, um das Ausführungstimeout festzulegen.

⚠ Important

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

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

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

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

Voraussetzungen

1. Führen Sie die Schritte unter [Einrichtung](#) aus.
2. Erstellen Sie ein Projekt in Device Farm. Befolgen Sie die Anweisungen unter [Erstellen Sie ein Projekt in AWS Device Farm](#), und kehren Sie dann zu dieser Seite zurück.

Legen Sie das Ausführungs-Timeout für ein Projekt fest

1. Melden Sie sich bei der Device Farm Farm-Konsole unter <https://console.aws.amazon.com/devicefarm> an.
2. Wählen Sie im Navigationsbereich Device Farm die Option Mobile Device Testing und dann Projects aus.

3. Wenn Sie bereits ein Projekt haben, wählen Sie es aus der Liste aus. Andernfalls wählen Sie Neues Projekt, geben Sie einen Namen für Ihr Projekt ein und wählen Sie dann Absenden.
4. Wählen Sie Project settings (Projekteinstellungen) aus.
5. Geben Sie auf der Registerkarte General (Allgemein) für Execution Timeout (Ausführungstimeout) einen Wert ein oder verwenden Sie den Schieberegler.
6. Wählen Sie Speichern.

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

Legen Sie das Ausführungs-Timeout für einen Testlauf fest

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

Simulieren Sie Netzwerkverbindungen und -bedingungen für Ihre AWS Device Farm Farm-Läufe

Sie können Network Shaping verwenden, um Netzwerkverbindungen und -bedingungen zu simulieren, während Sie Ihre Android-, iOS-, FireOS- und Web-Apps in Device Farm testen. So können Sie Ihre App z. B. in suboptimalen Netzwerkbedingungen testen.

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

Themen

- [Richten Sie Network Shaping ein, wenn Sie einen Testlauf planen](#)
- [Erstellen Sie ein Netzwerkprofil](#)
- [Ändern Sie die Netzwerkbedingungen während des Tests](#)

Richten Sie Network Shaping ein, wenn Sie einen Testlauf planen

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

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

Falls Sie noch keine Projekte besitzen, finden Sie unter [Erstellen Sie ein Projekt in AWS Device Farm](#) weitere Informationen.

2. Wählen Sie Ihre Anwendung und dann **Weiter** aus.
3. Konfigurieren Sie Ihren Test und wählen Sie dann **Weiter**.
4. Wählen Sie Ihre Geräte aus und wählen Sie dann **Weiter**.
5. Wählen Sie im Abschnitt **Standort- und Netzwerkeinstellungen** ein Netzwerkprofil aus, oder wählen Sie **Netzwerkprofil erstellen**, um Ihr eigenes zu erstellen.

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

Erstellen Sie ein Netzwerkprofil

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

1. Wählen Sie Netzwerkprofil erstellen.

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. Geben Sie einen Namen und die Einstellungen für Ihr Netzwerkprofil ein.
3. Wählen Sie Erstellen.

4. Beenden Sie das Erstellen Ihres Testlaufs und starten Sie den Durchlauf.

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

Network profiles						
Name	Bandwidth (bps)	Delay (ms)	Jitter (ms)	Loss (%)	Description	
<input type="radio"/>	▲ 104857600 ▼ 1048576	▲ 0 ▼ 0	▲ 0 ▼ 0	▲ 0 ▼ 0	-	<input type="checkbox"/>
<input type="radio"/>	▲ 104857600 ▼ 1048576	▲ 0 ▼ 0	▲ 0 ▼ 0	▲ 0 ▼ 0	-	<input type="checkbox"/>
<input type="radio"/>	▲ 104857600 ▼ 1048576	▲ 0 ▼ 0	▲ 0 ▼ 0	▲ 0 ▼ 0	-	<input type="checkbox"/>

Ändern Sie die Netzwerkbedingungen während des Tests

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

Stoppen Sie einen Lauf in AWS Device Farm

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

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

Themen

- [Stoppen Sie einen Lauf \(Konsole\)](#)
- [Stoppen Sie einen Lauf \(\)AWS CLI](#)
- [Stoppen Sie einen Lauf \(API\)](#)

Stoppen Sie einen Lauf (Konsole)

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

2. Wählen Sie im Navigationsbereich Device Farm die Option Mobile Device Testing und dann Projects aus.
3. Wählen Sie das Projekt aus, für das Sie einen aktiven Testlauf haben.
4. Wählen Sie auf der Seite Automatisierte Tests den Testlauf aus.

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

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

< 1 > ⓘ

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

5. Wählen Sie Stop run (Testlauf stoppen).

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

⚠ Important

Wenn ein Test bereits ausgeführt wurde, kann Device Farm ihn nicht beenden. Wenn ein Test läuft, stoppt Device Farm den Test. Die Gesamtzahl der Minuten, für die Sie Gebühren zahlen müssen, wird im Bereich Devices (Geräte) angezeigt. Darüber hinaus werden Ihnen auch die Gesamtminuten in Rechnung gestellt, die Device Farm benötigt,

um die Setup Suite und die Teardown Suite auszuführen. Weitere Informationen finden Sie unter [Device Farm – Preise](#).

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

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

Stoppen Sie einen Lauf ()AWS CLI

Sie können den folgenden Befehl ausführen, um den angegebenen Testlauf zu stoppen, wobei *myARN* der Amazon-Ressourcenname (ARN) des Testlaufs ist.

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

Die Ausgabe sollte folgendermaßen oder ähnlich aussehen:

```
{
  "run": {
    "status": "STOPPING",
    "name": "Name of your run",
    "created": 1458329687.951,
    "totalJobs": 7,
    "completedJobs": 5,
    "deviceMinutes": {
      "unmetered": 0.0,
      "total": 0.0,
      "metered": 0.0
    },
    "platform": "ANDROID_APP",
    "result": "PENDING",
    "billingMethod": "METERED",
    "type": "BUILTIN_EXPLORER",
```

```
    "arn": "myARN",
    "counters": {
      "skipped": 0,
      "warned": 0,
      "failed": 0,
      "stopped": 0,
      "passed": 0,
      "errored": 0,
      "total": 0
    }
  }
}
```

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

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

```
}  
  ]  
}
```

Informationen zur Verwendung von Device Farm mit dem AWS CLI finden Sie unter [AWS CLI-Referenz](#).

Stoppen Sie einen Lauf (API)

- Rufen Sie den [StopRun](#)Vorgang zum Testlauf auf.

Hinweise zur Verwendung der Device Farm API finden Sie unter [Gerätefarm wird automatisiert](#).

Eine Liste der Läufe in AWS Device Farm anzeigen

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

Themen

- [Eine Liste der Läufe anzeigen \(Konsole\)](#)
- [Eine Liste von Läufen anzeigen \(AWS CLI\)](#)
- [Eine Liste der Läufe anzeigen \(API\)](#)

Eine Liste der Läufe anzeigen (Konsole)

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

Tip

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

Eine Liste von Läufen anzeigen (AWS CLI)

- Führen Sie den Befehl [list-runs](#) aus.

Um Informationen zu einer einzigen Ausführung anzuzeigen, führen Sie den Befehl [get-run](#) aus.

Informationen zur Verwendung von Device Farm mit dem AWS CLI finden Sie unter [AWS CLI-Referenz](#).

Eine Liste der Läufe anzeigen (API)

- Rufen Sie die [ListRuns](#)-API auf.

Um Informationen zu einer einzigen Ausführung anzuzeigen, rufen Sie die [GetRun](#)-API auf.

Informationen zur Device Farm API finden Sie unter [Gerätefarm wird automatisiert](#).

Erstellen Sie einen Gerätepool in AWS Device Farm

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

Themen

- [Voraussetzungen](#)
- [Erstellen Sie einen Gerätepool \(Konsole\)](#)
- [Erstellen Sie einen Gerätepool \(AWS CLI\)](#)
- [Erstellen Sie einen Gerätepool \(API\)](#)

Voraussetzungen

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

Erstellen Sie einen Gerätepool (Konsole)

1. Wählen Sie auf der Seite „Geräte auswählen“ die Option Gerätepool erstellen aus.
2. Geben Sie unter Name einen Namen ein, an dem Sie den Gerätepool leicht erkennen können.
3. Geben Sie unter Description (Beschreibung) eine Beschreibung ein, an der Sie den Gerätepool leicht erkennen können.
4. Wenn Sie eine oder mehrere Auswahlkriterien für die Geräte in diesem Gerätepool verwenden möchten, führen Sie die folgenden Schritte aus:
 - a. Wählen Sie Dynamischen Gerätepool erstellen.
 - b. Wählen Sie Regel hinzufügen aus.
 - c. Wählen Sie für Feld (erste Dropdownliste) eine der folgenden Optionen aus:
 - Um Geräte nach ihrem Herstellernamen anzuzeigen, wählen Sie Gerätehersteller aus.
 - Um Geräte nach ihrem Typwert einzubeziehen, wählen Sie „Formfaktor“.
 - d. Wählen Sie für Operator (zweite Dropdownliste) die Option EQUALS aus, um Geräte einzubeziehen, bei denen der Feldwert dem Wertwert entspricht.
 - e. Geben Sie unter Wert (dritte Dropdownliste) den Wert ein, den Sie für die Werte Feld und Operator angeben möchten, oder wählen Sie ihn aus. Wenn Sie Platform (Plattform) für Field (Feld) auswählen, sind die einzigen verfügbaren Auswahlmöglichkeiten ANDROID und IOS. Wenn Sie für Field die Option Formfaktor wählen, sind ebenfalls nur die Optionen PHONE und TABLET verfügbar.
 - f. Um eine weitere Regel hinzuzufügen, wählen Sie Regel hinzufügen aus.
 - g. Um eine Regel zu löschen, wählen Sie das Symbol X neben der Regel aus.

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

5. Wenn Sie einzelne Geräte manuell ein- oder ausschließen möchten, gehen Sie wie folgt vor:
 - a. Wählen Sie Statischen Gerätepool erstellen aus.

- b. Aktivieren oder deaktivieren Sie das Kästchen neben jedem Gerät. Sie können die Kontrollkästchen nur aktivieren oder deaktivieren, wenn keine Regeln angegeben wurden.
6. Wenn Sie alle angezeigten Geräte ein- oder ausschließen möchten, aktivieren oder deaktivieren Sie das Kontrollkästchen in der Spaltenüberschriftenzeile der Liste.

 **Important**

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

7. Wählen Sie Erstellen.

Erstellen Sie einen Gerätepool (AWS CLI)

- Führen Sie den Befehl [create-device-pool](#) aus.

Informationen zur Verwendung von Device Farm mit dem AWS CLI finden Sie unter [AWS CLI-Referenz](#).

Erstellen Sie einen Gerätepool (API)

- Rufen Sie die [CreateDevicePool](#)-API auf.

Hinweise zur Verwendung der Device Farm API finden Sie unter [Gerätefarm wird automatisiert](#).

Analysieren der Ergebnisse in AWS Device Farm

In der Standardtestumgebung können Sie die Device Farm Farm-Konsole verwenden, um Berichte für jeden Test in Ihrem Testlauf anzuzeigen.

Device Farm sammelt auch andere Artefakte wie Dateien, Protokolle und Bilder, die Sie herunterladen können, wenn Ihr Testlauf abgeschlossen ist.

Themen

- [Arbeiten mit Testberichten in Device Farm](#)
- [Arbeiten mit Artefakten in Device Farm](#)

Arbeiten mit Testberichten in Device Farm

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

Themen

- [Voraussetzungen](#)
- [Testergebnisse verstehen](#)
- [Anzeigen von Berichten](#)

Voraussetzungen

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

1. Um einen Testlauf zu erstellen, befolgen Sie die Anweisungen unter [Erstellen Sie einen Testlauf in Device Farm](#) und kehren Sie dann zu dieser Seite zurück.
2. Stellen Sie sicher, dass der Testlauf abgeschlossen ist. Während Ihres Testlaufs zeigt die Device Farm Farm-Konsole ein Symbol  für laufende Läufe an. Jedes Gerät, das gerade ausgeführt wird, beginnt ebenfalls mit dem Symbol „Ausstehend“ und wechselt dann zu Beginn des Tests zum  Symbol „Wird ausgeführt“. Nach Abschluss jedes Tests wird neben dem Gerätenamen ein Testergebnissymbol angezeigt. Wenn alle Tests abgeschlossen sind, ändert sich das Symbol für ausstehende Tests neben dem Testlauf in ein Testergebnissymbol. Weitere Informationen finden Sie unter [Testergebnisse verstehen](#).

Testergebnisse verstehen

In der Device Farm Farm-Konsole werden Symbole angezeigt, mit denen Sie den Status Ihres abgeschlossenen Testlaufs schnell beurteilen können.

Themen

- [Ergebnisse für einen einzelnen Test melden](#)
- [Ergebnisse für mehrere Tests werden gemeldet](#)

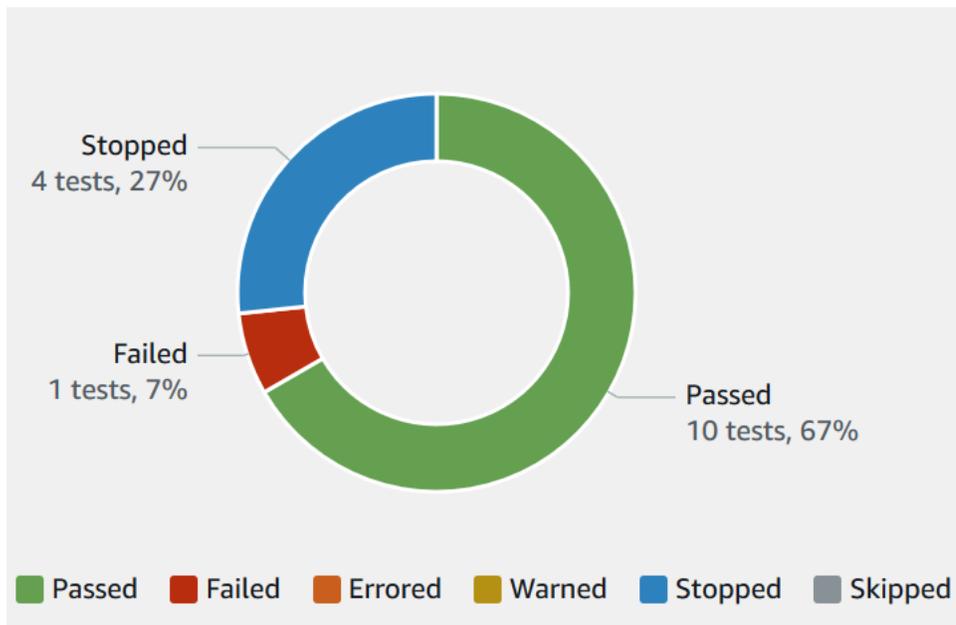
Ergebnisse für einen einzelnen Test melden

Für Berichte, die einen einzelnen Test beschreiben, zeigt Device Farm ein Symbol an:

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

Ergebnisse für mehrere Tests werden gemeldet

Wenn Sie einen abgeschlossenen Lauf wählen, zeigt Device Farm ein zusammenfassendes Diagramm mit den Testergebnissen an.



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

Diagramme sind immer farblich gekennzeichnet und beschriftet.

Anzeigen von Berichten

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

Themen

- [Sehen Sie sich die Seite mit der Zusammenfassung des Testlaufs an](#)
- [Sehen Sie sich einzigartige Problembereiche an](#)
- [Geräteberichte anzeigen](#)
- [Berichte der Testsuite anzeigen](#)
- [Anzeigen von Testberichten](#)
- [Leistungsdaten für ein Problem, ein Gerät, eine Suite oder einen Test in einem Bericht anzeigen](#)
- [Protokollinformationen für ein Problem, ein Gerät, eine Suite oder einen Test in einem Bericht anzeigen](#)

Sehen Sie sich die Seite mit der Zusammenfassung des Testlaufs an

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

2. Wählen Sie im Navigationsbereich Mobile Device Testing und dann Projects aus.
3. Wählen Sie in der Liste der Projekte das Projekt für den Testlauf aus.

 Tip

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

4. Wählen Sie einen abgeschlossenen Testlauf zum Anzeigen seiner Berichtsübersichtseite aus.
5. Die Testlauf-Übersichtsseite zeigt eine Übersicht Ihrer Testergebnisse an.
 - Im Bereich Unique problems (Eindeutige Probleme) werden eindeutige Warnungen und Fehler aufgeführt. Um eindeutige Probleme anzuzeigen, befolgen Sie die Anweisungen unter [Sehen Sie sich einzigartige Problembenachrichtigungen an](#).
 - Im Bereich Devices (Geräte) wird für jedes Gerät die Gesamtanzahl der Tests nach Ergebnissen angezeigt.

Devices	Unique problems	Screenshots	Parsing result	
<p>Devices</p> <p>Find device by status, device name, or OS</p> <p>< 1 > </p>				
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 diesem Beispiel gibt es mehrere Geräte. Im ersten Tabelleneintrag meldet das Google Pixel 4 XL-Gerät mit Android Version 10 drei erfolgreiche Tests, deren Ausführung 02:36 Minuten gedauert hat.

Um die Ergebnisse nach Gerät anzuzeigen, befolgen Sie die Anweisungen unter [Geräteberichte anzeigen](#).

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

Sehen Sie sich einzigartige Problemlisten an

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

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

Im Abschnitt Ergebnis wird das Ergebnis des Tests angezeigt. Der Status wird als Ergebnissymbol dargestellt. Weitere Informationen finden Sie unter [Ergebnisse für einen einzelnen Test melden](#).

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

Auf der Registerkarte Leistung werden Informationen zu allen Leistungsdaten angezeigt, die Device Farm während des Tests generiert hat. Sie können diese Leistungsdaten anzeigen, indem Sie die Anweisungen unter [Leistungsdaten für ein Problem, ein Gerät, eine Suite oder einen Test in einem Bericht anzeigen](#) befolgen.

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

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

Geräteberichte anzeigen

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

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

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

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

Um die vollständigen Ergebnisse nach Suiten aufgeschlüsselt anzuzeigen, folgen Sie den Anweisungen unter [Berichte der Testsuite anzeigen](#).

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

Im Abschnitt Leistung werden Informationen zu allen Leistungsdaten angezeigt, die Device Farm während der Ausführung für das Gerät generiert hat. Sie können diese Leistungsdaten anzeigen, indem Sie die Anweisungen unter [Leistungsdaten für ein Problem, ein Gerät, eine Suite oder einen Test in einem Bericht anzeigen](#) befolgen.

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

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

Berichte der Testsuite anzeigen

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

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

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

In der Tabelle wird in der Spalte Testergebnisse das Ergebnis angezeigt. Diese Daten haben auch eine grafische Komponente. Weitere Informationen finden Sie unter [Ergebnisse für mehrere Tests werden gemeldet](#).

Folgen Sie den Anweisungen unter, um die vollständigen Testergebnisse nach Tests einzusehen [Anzeigen von Testberichten](#).

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

Im Abschnitt Leistung werden Informationen zu allen Leistungsdaten angezeigt, die Device Farm während der Ausführung der Suite generiert hat. Sie können diese Leistungsdaten anzeigen, indem Sie die Anweisungen unter [Leistungsdaten für ein Problem, ein Gerät, eine Suite oder einen Test in einem Bericht anzeigen](#) befolgen.

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

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

Anzeigen von Testberichten

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

Im Abschnitt Ergebnis wird das Ergebnis des Tests angezeigt. Der Status wird als Ergebnissymbol dargestellt. Weitere Informationen finden Sie unter [Ergebnisse für einen einzelnen Test melden](#).

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

Auf der Registerkarte Leistung werden Informationen zu allen Leistungsdaten angezeigt, die Device Farm während des Tests generiert hat. Sie können diese Leistungsdaten anzeigen,

indem Sie die Anweisungen unter [Leistungsdaten für ein Problem, ein Gerät, eine Suite oder einen Test in einem Bericht anzeigen](#) befolgen.

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

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

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

Note

Device Farm sammelt derzeit nur Geräteleistungsdaten für Android-Geräte.

Auf der Registerkarte Leistung werden die folgenden Informationen angezeigt:

- Das CPU-Diagramm zeigt den prozentualen Anteil der CPU, den die App während des ausgewählten Problems, Geräts, der Suite oder des Tests auf einem einzelnen Kern verwendet hat (entlang der vertikalen Achse) im Zeitverlauf (entlang der horizontalen Achse).

Auf der vertikalen Achse wird der Prozentsatz von 0 % bis zum höchsten aufgezeichneten Prozentsatz angezeigt.

Dieser Prozentsatz kann 100 % überschreiten, wenn die App mehr als einen Kern verwendet hat. Wenn z. B. drei Kerne zu 60 % genutzt werden, wird ein Prozentsatz von 180 % angezeigt.

- Das Speicherdiagramm zeigt die Anzahl an MB, die die App während des ausgewählten Problems, Geräts, der Suite oder des Tests (entlang der vertikalen Achse) im Zeitverlauf (entlang der horizontalen Achse) verwendet hat.

Auf der vertikalen Achse wird die Arbeitsspeicherbelegung von 0 MB bis zur höchsten aufgezeichneten Belegung angezeigt.

- Das Diagramm Threads zeigt die Anzahl der verwendeten Threads für das ausgewählte Problem, Gerät, die Reihe oder den Test (vertikale Achse) im Zeitverlauf (horizontale Achse) an.

Die vertikale Achse wird in der Anzahl der Threads ausgedrückt, von Null Threads bis zur maximalen Anzahl aufgezeichneter Threads.

In allen Fällen wird auf der horizontalen Achse die Zeit vom Beginn bis zum Ende des Laufs in Sekunden für das ausgewählte Problem, Gerät, die Reihe oder den Test angezeigt.

Sie können die Informationen zu einem bestimmten Datenpunkt anzeigen, indem Sie das entsprechende Diagramm bei der gewünschten Sekunde auf der horizontalen Achse pausieren.

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

Im Abschnitt Protokolle werden die folgenden Informationen angezeigt:

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

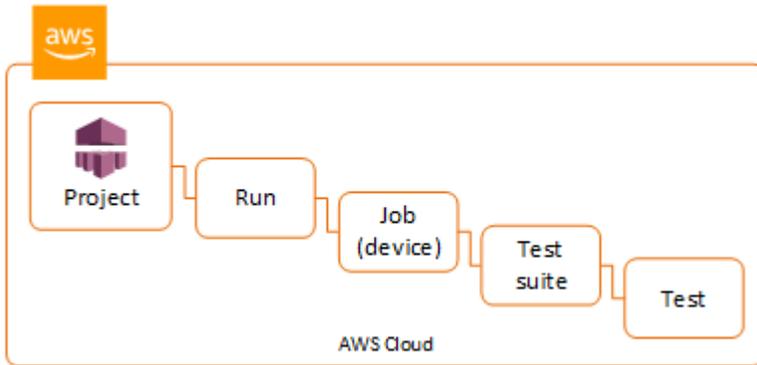
- Internal (Intern)
 - Notice (Hinweis)
 - Passed
 - Skipped
 - Angehalten
 - Verbose
 - Warned (Mit Warnungen)
 - Warnung
- Tag steht für beliebige Metadaten für den Protokolleintrag. Der Android-Systemlog (logcat) kann damit z. B. beschreiben, welcher Teil des Systems den Protokolleintrag erstellt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

So zeigen Sie nur einen Teil der Informationen an:

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

Um alle Protokollinformationen für das Gerät herunterzuladen, einschließlich aller Suiten und Tests, die Sie ausgeführt haben, wählen Sie Protokolle herunterladen.

Arbeiten mit Artefakten in Device Farm



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

Sie können die während des Testlaufs erstellten Artefakte herunterladen:

Dateien

Während des Testlaufs generierte Dateien, einschließlich Device Farm Farm-Berichten. Weitere Informationen finden Sie unter [Arbeiten mit Testberichten in Device Farm](#).

Logs (Protokolle)

Ausgabe aus jedem Test im Testlauf.

Screenshots

Bildschirmbilder, die für jeden Test im Testlauf aufgezeichnet wurden.

Verwendung von Artefakten (Konsole)

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

Weitere Informationen zum Herunterladen von Artefakten in einer benutzerdefinierten Testumgebung finden Sie unter [Verwendung von Artefakten in einer benutzerdefinierten Testumgebung](#).

Verwendung von Artefakten (AWS CLI)

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

Themen

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

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

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

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

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

Beispiel:

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

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

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

```

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

```

Schritt 2: Listet eure Artefakte auf

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

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

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

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

Die Antwort enthält eine Download-URL für jedes Artefakt.

```
{
```

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

Schritt 3: Laden Sie Ihre Artefakte herunter

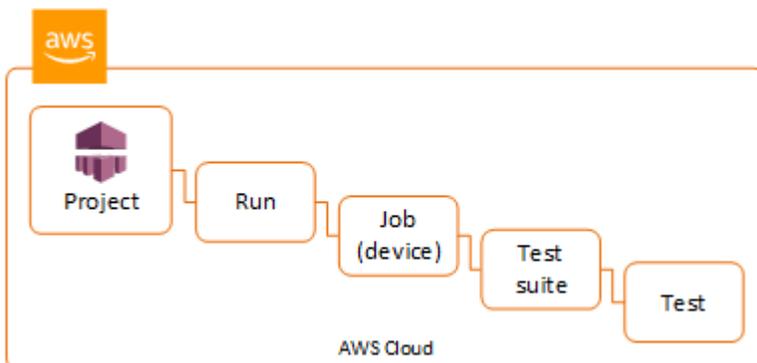
- Laden Sie Ihr Artefakt mithilfe der URL aus dem vorherigen Schritt herunter. Dieses Beispiel verwendet curl zum Herunterladen einer Android Appium Java-Ausgabedatei:

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

Verwenden von Artefakten (API)

Die Device Farm [ListArtifacts](#) API-Methode gibt eine Liste von Artefakten wie Dateien, Screenshots und Protokollen zurück. Jedes Artefakt verfügt über eine URL, sodass Sie die Datei herunterladen können.

Verwendung von Artefakten in einer benutzerdefinierten Testumgebung



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

Sie können diese während des Testlaufs erstellten Artefakte herunterladen:

Ausgabe der Testspezifikation

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

Kundenartefakte

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

Shell-Skript der Test-Spezifikation

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

Test-Spezifikationsdatei

Die im Testlauf verwendete YAML-Datei.

Weitere Informationen finden Sie unter [Arbeiten mit Artefakten in Device Farm](#).

Taggen von AWS Device Farm-Ressourcen

AWS Device Farm funktioniert mit der [AWS Tagging-API](#) für Ressourcengruppen. Mit dieser API können Sie Ressourcen in Ihrem AWS-Konto mit Tags verwalten. Sie können Ressourcen, wie Projekte und Testläufe, Tags hinzufügen.

Sie können Tags verwenden, um:

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

Weitere Informationen zum Markieren von Ressourcen finden Sie im Whitepaper [Bewährte Tagging-Methoden](#).

Themen

- [Markieren von Ressourcen](#)
- [Ressourcen nach Tag nachschlagen](#)
- [Entfernen von Tags von Ressourcen](#)

Markieren von Ressourcen

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

Verwenden Sie zum Taggen einer Ressource den Vorgang [TagResources](#) vom `resourcegroupstaggingapi`-Endpunkt aus. Dieser Vorgang generiert eine Liste von ARNs aus unterstützten Diensten und eine Liste von Schlüssel-Wert-Paaren. Der -Wert ist optional. Eine leere Zeichenfolge gibt an, dass für dieses Tag kein Wert vorhanden sein sollte. Im folgenden Python-Beispiel wird eine Reihe von Projekt-ARNs mit dem Tag `build-config` und dem Wert `release` versehen:

```
import boto3

client = boto3.client('resourcegroupstaggingapi')

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

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

Ressourcen nach Tag nachschlagen

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

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

Weitere Informationen finden Sie in der [API-Referenz der AWS-Ressourcengruppen-Markierung](#).

Das folgende Beispiel sucht nach Device Farm-Desktop-Browser-Testsitzungen (`devicefarm:testgrid-session`Ressourcen) mit dem Tagstack die den Wert `habenproduction`:

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

Entfernen von Tags von Ressourcen

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

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

Arbeiten mit Testtypen in AWS Device Farm

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

Frameworks testen

Device Farm unterstützt die folgenden Frameworks für mobile Automatisierungstests:

Frameworks zum Testen von Android-Anwendungen

- [Arbeiten mit Appium und AWS Device Farm](#)
- [Arbeiten mit Instrumentierung für Android und AWS Device Farm](#)

Frameworks zum Testen von iOS-Anwendungen

- [Arbeiten mit Appium und AWS Device Farm](#)
- [Arbeiten mit XCTest für iOS und AWS Device Farm](#)
- [XCTest-UI](#)

Frameworks zum Testen von Webanwendungen

Webanwendungen werden durch Appium unterstützt. Weitere Informationen dazu, wie Sie Ihre Tests mit Appium verwenden, finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

Frameworks in einer benutzerdefinierten Testumgebung

Device Farm bietet keine Unterstützung für die Anpassung der Testumgebung für das XCTest-Framework. Weitere Informationen finden Sie unter [Arbeiten mit benutzerdefinierten Testumgebungen](#).

Unterstützung der Appium-Version

Für Tests, die in einer benutzerdefinierten Umgebung ausgeführt werden, unterstützt Device Farm Appium Version 1. Weitere Informationen finden Sie unter [Testumgebungen](#).

Integrierte Testtypen

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

- [Eingebaut: Fuzz \(Android und iOS\)](#)

Arbeiten mit Appium und AWS Device Farm

In diesem Abschnitt wird beschrieben, wie Sie Ihre Appium-Tests konfigurieren, verpacken und auf Device Farm hochladen. Appium ist ein Open-Source-Tool zur Automatisierung nativer und mobiler Webanwendungen. Weitere Informationen finden Sie unter [Einführung in Appium](#) auf der Appium-Website.

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

Versionsunterstützung

Der Support verschiedener Frameworks und Programmiersprachen hängt von der verwendeten Sprache ab.

Device Farm unterstützt alle Appium 1.x- und 2.x-Serverversionen. Für Android können Sie eine beliebige Appium-Hauptversion mit wählen. `devicefarm-cli` Um beispielsweise Appium Server Version 2 zu verwenden, fügen Sie die folgenden Befehle zu Ihrer YAML-Datei mit Testspezifikationen hinzu:

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

Für iOS können Sie mit den npm Befehlen `avm` oder bestimmte Appium-Versionen auswählen. Um beispielsweise den `avm` Befehl zu verwenden, um die Appium-Serverversion auf 2.1.2 zu setzen, fügen Sie die folgenden Befehle zu Ihrer YAML-Datei mit Testspezifikationen hinzu:

```
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
```

Verwenden Sie den npm Befehl, um die neueste Version von Appium 2 zu verwenden, und fügen Sie die folgenden Befehle zu Ihrer YAML-Datei mit Testspezifikationen hinzu:

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

Weitere Informationen zu `devicefarm-cli` oder anderen CLI-Befehlen finden Sie in der [AWS-CLI-Referenz](#).

Um alle Funktionen des Frameworks wie Anmerkungen zu nutzen, wählen Sie eine benutzerdefinierte Testumgebung und laden Sie mithilfe der AWS-CLI oder der Device Farm Konsole eine benutzerdefinierte Testspezifikation hoch.

Themen

- [Konfigurieren Sie Ihr Appium-Testpaket](#)
- [Erstellen Sie eine komprimierte Testpaketdatei](#)
- [Laden Sie Ihr Testpaket auf Device Farm hoch](#)
- [Machen Sie Screenshots Ihrer Tests \(optional\)](#)

Konfigurieren Sie Ihr Appium-Testpaket

Anhand der folgenden Anweisungen können Sie Ihr Testpaket konfigurieren.

Java (JUnit)

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

```
<groupId>com.acme</groupId>
<artifactId>acme-myApp-appium</artifactId>
```

```
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>
```

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

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

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

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

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

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

```

    </executions>
  </plugin>

```

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

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

```

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

```

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

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

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

Note

Wenn Sie eine Fehlermeldung erhalten, dass in Version 1.3 Anmerkungen nicht unterstützt werden, fügen Sie Folgendes in `pom.xml` ein:

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

Java (TestNG)

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

```
<groupId>com.acme</groupId>
```

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

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

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

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

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

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

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

```

    </execution>
  </executions>
</plugin>

```

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

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

```

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

```

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

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

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

Note

Wenn Sie eine Fehlermeldung erhalten, dass in Version 1.3 Anmerkungen nicht unterstützt werden, fügen Sie Folgendes in `pom.xml` ein:

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

Node.JS

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

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

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

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

1. Stellen Sie sicher, dass nvm vorhanden ist.

```
command -v nvm
```

Sie sollten als Ausgabe nvm sehen.

Weitere Informationen finden Sie unter [nvm on](#). GitHub

2. Führen Sie diesen Befehl aus, um Node.js zu installieren:

```
nvm install node
```

Sie können eine bestimmte Version von Node.js angeben:

```
nvm install 11.4.0
```

3. Stellen Sie sicher, dass die richtige Version von Node verwendet wird:

```
node -v
```

4. Installieren Sie npm-bundle global:

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

Python

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

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

Tip

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

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

```
$ pip install pytest
```

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

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

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

```
$ find tests/
```

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

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

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

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

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

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

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

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

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

Ruby

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

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

Verwenden Sie dieses Befehlszeilen-Tool, wenn Sie Ihre Tests entwickeln und Testpakete erstellen, um zu vermeiden, dass unnötige Abhängigkeiten in Ihr Testpaket eingeschlossen werden.

- Ruby
- Bundler (Dieses Gem wird in der Regel mit Ruby installiert.)

1. Installieren Sie die erforderlichen Schlüssel, RVM und Ruby. Anweisungen finden Sie unter [Installing RVM](#) auf der RVM-Website.

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

 Note

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

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

```
command -v rvm
```

Sie sollten als Ausgabe `rvm` sehen.

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

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

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

```
ruby -v
```

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

```
bundle config specific_platform true
```

5. Aktualisieren Sie Ihre `.lock`-Datei, um die Plattformen hinzuzufügen, die für die Ausführung von Tests benötigt werden.
 - Wenn Sie Tests für die Ausführung auf Android-Geräten kompilieren, führen Sie diesen Befehl aus, um das Gemfile so zu konfigurieren, dass es Abhängigkeiten für den Android-Testhost verwendet:

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

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

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

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

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

Erstellen Sie eine komprimierte Testpaketdatei

Warning

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

Bündeln Sie nun Ihre Tests für die Device Farm.

Java (JUnit)

Erstellen und verpacken Sie Ihre Tests:

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

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

Java (TestNG)

Erstellen und verpacken Sie Ihre Tests:

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

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

Node.JS

1. Überprüfen Sie Ihr Projekt.

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

2. Führen Sie diesen Befehl aus, um Ihre lokalen Abhängigkeiten zu installieren.

```
npm install
```

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

Note

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

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

```
npm-bundle
```

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

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

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

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

Python

Python 2

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

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

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

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

Python 3

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

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

Ruby

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

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

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

```
rvm gemset use myGemset
```

3. Überprüfen Sie den Quellcode.

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

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

```
bundle install
```

Note

Zu diesem Zeitpunkt sollten Sie in der Lage sein, Ihre Tests lokal auszuführen. Mit diesem Befehl können Sie einen lokalen Test ausführen:

```
bundle exec $test_command
```

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

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

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

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

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

Laden Sie Ihr Testpaket auf Device Farm hoch

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

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

Wenn Sie bereits ein Projekt haben, können Sie es auswählen, um Ihre Tests darauf hochzuladen.

4. Öffnen Sie Ihr Projekt und wählen Sie Create a new run (Neuen Lauf erstellen).
5. Für native Android- und iOS-Tests

Wählen Sie auf der Seite „Anwendung auswählen“ die Option „Mobile App“ und anschließend „Datei auswählen“ aus, um das verteilbare Paket Ihrer Anwendung hochzuladen.

Note

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

Für Tests von mobilen Webanwendungen

Wählen Sie auf der Seite „Anwendung auswählen“ die Option Web-App aus.

6. Geben Sie Ihrem Test einen entsprechenden Namen. Dieser kann eine beliebige Kombination aus Leerzeichen oder Satzzeichen enthalten.
7. Wählen Sie Weiter aus.
8. Wählen Sie auf der Seite „Konfigurieren“ im Abschnitt „Setup Test Framework“ die Option **Appium-Sprache** und dann Datei auswählen aus.
9. Navigieren Sie zu der ZIP-Datei, die Ihre Tests enthält, und wählen Sie diese aus. Die ZIP-Datei muss dem Format entsprechen, das unter [Konfigurieren Sie Ihr Appium-Testpaket](#) beschrieben wird.
10. Wählen Sie Test in einer benutzerdefinierten Umgebung ausführen aus. Diese Ausführungsumgebung ermöglicht die vollständige Kontrolle über die Einrichtung, den Teardown und den Aufruf von Tests sowie die Auswahl bestimmter Versionen von Laufzeiten und des Appium-Servers. Sie können Ihre benutzerdefinierte Umgebung über die Testspezifikationsdatei konfigurieren. Weitere Informationen finden Sie unter [Arbeiten mit benutzerdefinierten Testumgebungen in AWS Device Farm](#).
11. Wählen Sie Weiter und folgen Sie dann den Anweisungen, um Geräte auszuwählen und die Ausführung zu starten. Weitere Informationen finden Sie unter [Erstellen Sie einen Testlauf in Device Farm](#).

Note

Device Farm ändert Appium-Tests nicht.

Machen Sie Screenshots Ihrer Tests (optional)

Sie können im Rahmen Ihrer Tests Screenshots erstellen.

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

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

Arbeiten mit Android-Tests in AWS Device Farm

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

Frameworks zum Testen von Android-Anwendungen

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

- [Arbeiten mit Appium und AWS Device Farm](#)
- [Arbeiten mit Instrumentierung für Android und AWS Device Farm](#)

Integrierte Testtypen für Android

Für Android-Geräte ist ein integrierter Testtyp verfügbar.

- [Eingebaut: Fuzz \(Android und iOS\)](#)

Arbeiten mit Instrumentierung für Android und AWS Device Farm

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

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

Themen

- [Was ist Instrumentierung?](#)
- [Laden Sie Ihre Android-Instrumentierungstests hoch](#)
- [Erstellen von Screenshots bei Android-Instrumentierungstests](#)
- [Zusätzliche Überlegungen zu Android-Instrumentierungstests](#)
- [Testanalyse im Standardmodus](#)

Was ist Instrumentierung?

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

Laden Sie Ihre Android-Instrumentierungstests hoch

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

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

Tip

Sie können die Suchleiste verwenden, um die Projektliste nach Namen zu filtern. Befolgen Sie die Anweisungen unter [Erstellen Sie ein Projekt in AWS Device Farm](#), um ein neues Projekt zu erstellen.

4. Wenn die Schaltfläche Create a new run (Einen neuen Testlauf erstellen) angezeigt wird, wählen Sie diese Option aus.
5. Wählen Sie auf der Seite „Anwendung auswählen“ die Option Datei auswählen aus.
6. Navigieren Sie zu der Datei mit Ihrer Android-Anwendung, und wählen Sie diese aus. Es muss sich dabei um eine APK-Datei handeln.
7. Wählen Sie Weiter aus.

8. Wählen Sie auf der Seite „Konfigurieren“ im Abschnitt Test-Framework einrichten die Option Instrumentation und dann Datei auswählen aus.
9. Navigieren Sie zu der APK-Datei, die Ihre Tests enthält, und wählen Sie diese aus.
10. Wählen Sie Weiter und führen Sie dann die verbleibenden Anweisungen aus, um Geräte auszuwählen und den Rechenlauf zu starten.

Erstellen von Screenshots bei Android-Instrumentierungstests

Sie können im Rahmen Ihrer Android-Instrumentierungstests Screenshots erstellen.

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

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

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

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

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

Zusätzliche Überlegungen zu Android-Instrumentierungstests

System Animations (Systemanimationen)

Gemäß der [Android-Dokumentation für Espresso-Tests](#) wird empfohlen, die Systemanimationen beim Testen auf echten Geräten auszuschalten. Device Farm deaktiviert automatisch die Einstellungen Window Animation Scale, Transition Animation Scale und Animator Duration Scale,

wenn es mit dem Instrumentationstest-Runner `UnitRunner` [android.support.test.runner.AndroidJUnit4](https://developer.android.com/reference/android/support/test/runner/AndroidJUnit4) ausgeführt wird.

Test Recorders (Test-Aufzeichnungen)

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

Testanalyse im Standardmodus

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

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

Um dies in einer lokalen Umgebung zu testen:

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

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

Arbeiten mit iOS-Tests in AWS Device Farm

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

Frameworks zum Testen von iOS-Anwendungen

Die folgenden Tests stehen für iOS-Geräte zur Verfügung.

- [Arbeiten mit Appium und AWS Device Farm](#)
- [Arbeiten mit XCTest für iOS und AWS Device Farm](#)
- [XCTest-UI](#)

Integrierte Testtypen für iOS

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

- [Eingebaut: Fuzz \(Android und iOS\)](#)

Arbeiten mit XCTest für iOS und AWS Device Farm

Mit Device Farm können Sie das XCTest-Framework verwenden, um Ihre App auf echten Geräten zu testen. Weitere Informationen zu XCTest finden Sie unter [Testgrundlagen](#) in Testen mit Xcode.

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

Themen

- [Erstellen Sie die Pakete für Ihren XCTestlauf](#)
- [Hochladen der Pakete für Ihren XCTest-Lauf auf Device Farm](#)

Erstellen Sie die Pakete für Ihren XCTestlauf

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

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

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

So generieren Sie die Build-Ausgabe für Ihre App:

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

So erstellen Sie das App-Paket:

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

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

6. Optional können Sie den Ordner Payload und die Datei .app darin löschen.

So erstellen Sie das XCTest-Paket:

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

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

Hochladen der Pakete für Ihren XCTest-Lauf auf Device Farm

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

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

Andernfalls wählen Sie im Device Farm Farm-Navigationsbereich die Option Mobile Device Testing und dann Projects aus.

3. Wählen Sie das Projekt aus, mit dem Sie den Test ausführen möchten.
4. Wählen Sie Create a new run (Neuen Lauf erstellen).
5. Wählen Sie auf der Seite „Anwendung auswählen“ die Option Mobile App aus.
6. Wählen Sie „Datei auswählen“.
7. Navigieren Sie zur .ipa-Datei für Ihre App und laden Sie sie hoch.

 Note

Ihr .ipa-Paket muss für Tests erstellt sein.

8. Nachdem der Upload abgeschlossen ist, wählen Sie Weiter.
9. Wählen Sie auf der Seite Konfigurieren im Abschnitt Setup Test Framework die Option XCTest aus. Wählen Sie dann Datei auswählen.
10. Navigieren Sie zu der .zip-Datei, die das XCTest-Paket für Ihre App enthält, und laden Sie es hoch.
11. Nachdem der Upload abgeschlossen ist, wählen Sie Weiter.
12. Führen Sie die restlichen Schritte der Projekterstellung durch. Sie wählen die Geräte für den Test und geben den gerätezustand an.
13. Nachdem Sie Ihren Lauf konfiguriert haben, wählen Sie auf der Seite Ausführung überprüfen und starten die Option Ausführung bestätigen und starten aus.

Device Farm führt Ihren Test aus und zeigt die Ergebnisse in der Konsole an.

Arbeiten mit dem XCTest UI-Testframework für iOS und AWS Device Farm

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

Themen

- [Was ist das XCTest UI-Testframework?](#)
- [Bereiten Sie Ihre iOS XCTest UI-Tests vor](#)
- [Laden Sie Ihre iOS XCTest UI-Tests hoch](#)
- [Screenshots in iOS XCTest UI-Tests machen](#)

Was ist das XCTest UI-Testframework?

Das XCTest-Framework für Benutzeroberflächen ist das neue Testframework, das mit Xcode 7 eingeführt wurde. Dieses XCTest-Framework für Benutzeroberflächen erweitert XCTest um Funktionen für das Testen von Benutzeroberflächen. Weitere Informationen finden Sie unter [User Interface Testing](#) in der iOS Developer Library.

Bereiten Sie Ihre iOS XCTest UI-Tests vor

Ihr iOS XCTest UI-Test-Runner-Paket muss in einer ordnungsgemäß formatierten .ipa-Datei enthalten sein.

Um eine IPA-Datei zu erstellen, platzieren Sie Ihr my-project-name UITest-Runner.app-Bundle in einem leeren Payload-Verzeichnis. Danach archivieren Sie das Payload-Verzeichnis in einer ZIP-Datei und ändern anschließend die Dateierweiterung in .ipa. Das *UITest-Runner.app-Paket wird von Xcode erstellt, wenn Sie Ihr Testprojekt erstellen. Es ist im Verzeichnis "Products" für das Projekt zu finden.

Laden Sie Ihre iOS XCTest UI-Tests hoch

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

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

Tip

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

Um ein Projekt zu erstellen, folgen Sie den Anweisungen unter [Erstellen Sie ein Projekt in AWS Device Farm](#)

4. Wenn die Schaltfläche Create a new run (Einen neuen Testlauf erstellen) angezeigt wird, wählen Sie diese Option aus.
5. Wählen Sie auf der Seite „Anwendung auswählen“ die Option Datei auswählen aus.
6. Navigieren Sie zu der Datei mit Ihrer iOS-Anwendung und wählen Sie diese aus. Es muss sich dabei um eine IPA-Datei handeln.

 Note

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

7. Wählen Sie Weiter aus.
8. Wählen Sie auf der Seite Konfigurieren im Abschnitt Setup Test Framework die Option XCTest UI und dann Datei auswählen aus.
9. Navigieren Sie zu der IPA-Datei, die den iOS-Runner für Ihren XCTest-Benutzeroberflächentest enthält, und wählen Sie diese aus.
10. Wählen Sie Weiter und befolgen Sie dann die verbleibenden Anweisungen, um die Geräte auszuwählen, auf denen Ihre Tests ausgeführt werden sollen, und starten Sie den Testlauf.

Screenshots in iOS XCTest UI-Tests machen

XCTest-Benutzeroberflächentests erstellen für jeden Schritt eines Tests automatisch einen Screenshot. Diese Screenshots werden in Ihrem Device Farm Farm-Testbericht angezeigt. Es ist kein zusätzlicher Code erforderlich.

Arbeiten mit Web-App-Tests in AWS Device Farm

Device Farm bietet Tests mit Appium für Webanwendungen. Weitere Informationen zum Einrichten Ihrer Appium-Tests auf Device Farm finden Sie unter [the section called “Appium”](#).

Regeln für Geräte mit und ohne Messgerät

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

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

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

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

Arbeiten mit integrierten Tests in AWS Device Farm

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

Integrierte Testtypen

Mit integrierten Tests können Sie Ihre Apps testen, ohne Skripts schreiben zu müssen.

- [Eingebaut: Fuzz \(Android und iOS\)](#)

Arbeiten mit dem integrierten Fuzz-Test für Device Farm

Device Farm bietet einen integrierten Fuzz-Testtyp.

Was ist der integrierte Fuzz-Test?

Der integrierte Fuzz-Test sendet zufallsgenerierte Benutzeroberflächen-Ereignisse an zu testende Geräte und erfasst dann die Ergebnisse in entsprechenden Berichten.

Verwenden Sie den integrierten Fuzz-Testtyp

Verwenden Sie die Device Farm Farm-Konsole, um den integrierten Fuzz-Test auszuführen.

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

 Tip

Sie können die Suchleiste verwenden, um die Projektliste nach Namen zu filtern. Befolgen Sie die Anweisungen unter [Erstellen Sie ein Projekt in AWS Device Farm](#), um ein neues Projekt zu erstellen.

4. Wenn die Schaltfläche Create a new run (Einen neuen Testlauf erstellen) angezeigt wird, wählen Sie diese Option aus.
5. Wählen Sie auf der Seite „Anwendung auswählen“ die Option Datei auswählen aus.
6. Navigieren Sie zu der Datei mit Ihrer Anwendung, für die Sie den integrierten Fuzz-Test ausführen möchten, und wählen Sie diese aus.
7. Wählen Sie Weiter aus.
8. Wählen Sie auf der Seite „Konfigurieren“ im Abschnitt „Setup Test Framework“ die Option Built-In: Fuzz aus.
9. Wenn eine oder mehrere der folgenden Einstellungen angezeigt wird, können Sie entweder die Standardwerte übernehmen oder eigene Werte angeben:
 - Event count (Ereignisanzahl): Geben Sie eine Zahl zwischen 1 und 10.000 ein, welche die Anzahl der Benutzeroberflächen-Ereignisse angibt, die für den auszuführenden Fuzz-Test generiert werden soll.
 - Event-Throttling: Geben Sie eine Zahl zwischen 0 und 1.000 an, die angibt, wie viele Millisekunden der Fuzz-Test warten muss, bevor das nächste Benutzeroberflächenereignis ausgeführt wird.
 - Randomizer seed (Randomisierungs-Seed): Geben Sie eine Zahl an, die vom auszuführenden Fuzz-Test als Startwert für die Randomisierung von Benutzeroberflächen-Ereignissen verwendet werden soll. Die Verwendung desselben Werts für aufeinander folgende Fuzz-Tests gewährleistet identische Ereignissequenzen.

10. Wählen Sie Weiter und führen Sie dann die verbleibenden Anweisungen aus, um Geräte auszuwählen und den Rechenlauf zu starten.

Arbeiten mit benutzerdefinierten Testumgebungen in AWS Device Farm

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

Zu den Vorteilen des benutzerdefinierten Modus im Gegensatz zum Standardmodus gehören:

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

Im Gegensatz dazu können Sie mit dem benutzerdefinierten Modus Ihre Testausführungsumgebung von Device Farm konsistent mit Ihrer lokalen Testumgebung in Einklang bringen.

Benutzerdefinierte Umgebungen werden mit einer YAML-formatierten Testspezifikationsdatei (Testspezifikation) konfiguriert. Device Farm bietet eine Standard-Testspezifikationsdatei für jeden unterstützten Testtyp, der unverändert verwendet oder angepasst werden kann. Anpassungen wie Testfilter oder Konfigurationsdateien können der Testspezifikation hinzugefügt werden. Bearbeitete Testspezifikationen können für zukünftige Testläufe gespeichert werden.

Weitere Informationen finden Sie unter [Hochladen einer benutzerdefinierten Testspezifikation mit der AWS CLI](#) und [Erstellen Sie einen Testlauf in Device Farm](#).

Themen

- [Syntax der Testspezifikation](#)
- [Beispiel für eine Testspezifikation](#)
- [Arbeiten mit der Amazon Linux 2-Testumgebung für Android-Tests](#)
- [Umgebungsvariablen](#)
- [Migrieren von Tests von einer Standard-Testumgebung zu einer benutzerdefinierten Testumgebung](#)
- [Erweiterung benutzerdefinierter Testumgebungen in Device Farm](#)

Syntax der Testspezifikation

Dies ist die YAML-Testspezifikationsstruktur:

```
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
```

Die Testspezifikation enthält Folgendes:

version

Entspricht der von Device Farm unterstützten Testspezifikationsversion. Die aktuelle Versionsnummer ist 0.1.

phases

Dieser Abschnitt enthält Gruppen gängiger Befehle, die während eines Testlaufs ausgeführt werden.

Die zulässigen Testphasennamen sind:

install

Optional.

Standardabhängigkeiten für Test-Frameworks, die von Device Farm unterstützt werden, sind bereits installiert. Diese Phase enthält zusätzliche Befehle, falls vorhanden, die Device Farm während der Installation ausführt.

pre_test

Optional.

Die Befehle, die ggf. vor Ihrem automatisierten Testlauf ausgeführt werden.

test

Optional.

Die Befehle, die während Ihrem automatisierten Testlauf ausgeführt werden. Wenn ein Befehl in der Testphase fehlschlägt, wird der Test als fehlgeschlagen gekennzeichnet.

post_test

Optional.

Die Befehle, die ggf. nach Ihrem automatisierten Testlauf ausgeführt werden.

artifacts

Optional.

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

Diese Testartefakte sind für jedes Gerät in Ihrem Testlauf verfügbar. Weitere Informationen zum Abrufen Ihrer Testartefakte finden Sie unter [Verwendung von Artefakten in einer benutzerdefinierten Testumgebung](#).

Important

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

Beispiel für eine Testspezifikation

Dies ist ein Beispiel für eine Device Farm YAML-Testspezifikation, die einen Appium Java TestNG-Testlauf konfiguriert:

```
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 XCUIest 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 XCUIest 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
```

Arbeiten mit der Amazon Linux 2-Testumgebung für Android-Tests

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

Der Amazon Linux 2-Testhost ist die neueste Android-Testumgebung und ersetzt das vorherige Ubuntu-basierte System. Mithilfe Ihrer Testspezifikationsdatei können Sie wählen, ob Sie Ihre Android-Tests in der Amazon Linux 2-Umgebung ausführen möchten.

Der Amazon Linux 2-Host bietet mehrere Vorteile:

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

Themen

- [Unterstützte Software](#)
- [Das devicefarm-cli-Tool](#)
- [Auswahl des Android-Testhosts](#)
- [Beispiel für eine Testspezifikationsdatei](#)
- [Migration zum Amazon Linux 2-Testhost](#)

Unterstützte Software

Auf dem Amazon Linux 2-Testhost sind viele der erforderlichen Softwarebibliotheken zur Unterstützung von Device Farm Farm-Testframeworks vorinstalliert, sodass beim Start eine einsatzbereite Testumgebung zur Verfügung steht. Für jede andere erforderliche Software können Sie die Testspezifikationsdatei so ändern, dass sie von Ihrem Testpaket aus installiert, aus dem Internet heruntergeladen oder auf private Quellen in Ihrer VPC zugegriffen wird (weitere Informationen finden Sie unter [VPC ENI](#)). Weitere Informationen finden Sie im Beispiel für die [Testspezifikationsdatei](#).

Die folgenden Softwareversionen sind derzeit auf dem Host verfügbar:

Softwarebibliothek	Softwareversion	Befehl zur Verwendung in Ihrer Testspezifikationsdatei
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>
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>

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>

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

Das `devicefarm-cli`-Tool

Der Amazon Linux 2-Testhost verwendet ein standardisiertes Versionsverwaltungstool, das `devicefarm-cli` zur Auswahl von Softwareversionen aufgerufen wird. Dieses Tool ist unabhängig vom Device Farm Test Host AWS CLI und nur auf dem Device Farm Test Host verfügbar. Mit `devicefarm-cli` können Sie zu einer beliebigen vorinstallierten Softwareversion auf dem Testhost wechseln. Dies bietet eine einfache Möglichkeit, Ihre Device Farm Farm-Testspezifikationsdatei im Laufe der Zeit zu verwalten, und bietet Ihnen einen vorhersehbaren Mechanismus, um Softwareversionen in future zu aktualisieren.

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

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

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

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

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

Um die Appium-Version von 1 auf 2 zu ändern:

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

Tip

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

Auswahl des Android-Testhosts

Warning

Der ältere Android-Testhost wird am 21. Oktober 2024 nicht mehr verfügbar sein. Beachten Sie, dass der Prozess für die Veralterung auf mehrere Termine aufgeteilt ist:

- Am 22. April 2024 werden Jobs von jedem neuen Konto an den aktualisierten Testhost weitergeleitet.
- Am 2. September 2024 müssen alle neuen oder geänderten Testspezifikationsdateien auf den aktualisierten Testhost abzielen.
- Am 21. Oktober 2024 können Jobs nicht mehr auf dem alten Testhost ausgeführt werden.

Stellen Sie Ihre Testspezifikationsdateien auf den `amazon_linux_2` Host ein, um Kompatibilitätsprobleme zu vermeiden.

Für Android-Tests benötigt Device Farm das folgende Feld in Ihrer Testspezifikationsdatei, um den Amazon Linux 2-Testhost auszuwählen:

```
android_test_host: amazon_linux_2 | legacy
```

Verwenden Sie `amazon_linux_2`, um Ihre Tests auf dem Amazon Linux 2-Testhost auszuführen:

```
android_test_host: amazon_linux_2
```

Erfahren Sie [hier](#) mehr über die Vorteile von Amazon Linux 2.

Device Farm empfiehlt, den Amazon Linux 2-Host für Android-Tests anstelle der Legacy-Host-Umgebung zu verwenden. Wenn Sie lieber die Legacy-Umgebung verwenden möchten, verwenden Sie diese, `legacy` um Ihre Tests auf dem Legacy-Testhost auszuführen:

```
android_test_host: legacy
```

Standardmäßig werden Testspezifikationsdateien ohne Testhostauswahl auf dem alten Testhost ausgeführt.

Veraltete Syntax

Im Folgenden finden Sie die veraltete Syntax für die Auswahl von Amazon Linux 2 in Ihrer Testspezifikationsdatei:

```
preview_features:  
  android_amazon_linux_2_host: true
```

Wenn Sie dieses Flag verwenden, werden Ihre Tests weiterhin auf Amazon Linux 2 ausgeführt. Wir empfehlen jedoch dringend, den Abschnitt `preview_features` Flags zu entfernen und ihn durch das neue `android_test_host` Feld zu ersetzen, um future Wartungsaufwand zu vermeiden.

⚠ Warning

Wenn Sie `android_test_host` sowohl die `android_amazon_linux_2_host` Flags als auch in Ihrer Testspezifikationsdatei verwenden, wird ein Fehler zurückgegeben. Es sollte nur einer verwendet werden; wir empfehlen `android_test_host`.

Beispiel für eine Testspezifikationsdatei

Der folgende Ausschnitt ist ein Beispiel für eine Device Farm Farm-Testspezifikationsdatei, die einen Appium NodeJS-Testlauf mit dem Amazon Linux 2-Testhost für Android konfiguriert:

```
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
```

Migration zum Amazon Linux 2-Testhost

Warning

Der ältere Android-Testhost wird am 21. Oktober 2024 nicht mehr verfügbar sein. Beachten Sie, dass der Prozess für die Veralterung auf mehrere Termine aufgeteilt ist:

- Am 22. April 2024 werden Jobs von jedem neuen Konto an den aktualisierten Testhost weitergeleitet.
- Am 2. September 2024 müssen alle neuen oder geänderten Testspezifikationsdateien auf den aktualisierten Testhost abzielen.
- Am 21. Oktober 2024 können Jobs nicht mehr auf dem alten Testhost ausgeführt werden.

Stellen Sie Ihre Testspezifikationsdateien auf den `amazon_linux_2` Host ein, um Kompatibilitätsprobleme zu vermeiden.

Um bestehende Tests vom alten Host auf den neuen Amazon Linux 2-Host zu migrieren, entwickeln Sie neue Testspezifikationsdateien, die auf Ihren bereits vorhandenen basieren. Es wird empfohlen, mit den neuen Standard-Testspezifikationsdateien für Ihre Testtypen zu beginnen. Migrieren Sie dann die relevanten Befehle von Ihrer alten Testspezifikationsdatei zur neuen und speichern Sie die alte Datei als Backup. Auf diese Weise können Sie die optimierte Standardspezifikation für den neuen Host nutzen und gleichzeitig Ihren vorhandenen Code wiederverwenden. Dadurch wird sichergestellt, dass Sie alle Vorteile des neuen Hosts nutzen können, der optimal für Ihre Tests konfiguriert ist, und gleichzeitig Ihre alte Testspezifikation als Referenz beibehalten, wenn Sie Befehle an die neue Umgebung anpassen.

Die folgenden Schritte können verwendet werden, um eine neue Amazon Linux 2-Testspezifikationsdatei zu erstellen und dabei Befehle aus Ihrer alten Testspezifikationsdatei wiederzuverwenden:

1. Melden Sie sich bei der Device Farm Farm-Konsole unter <https://console.aws.amazon.com/devicefarm> an.
2. Navigieren Sie zum Device Farm Farm-Projekt, das Ihre Automatisierungstests enthält.
3. Wählen Sie im Projekt Neuen Testlauf erstellen aus.
4. Wählen Sie eine zuvor verwendete App und ein Testpaket für Ihr Testframework aus.

5. Wählen Sie Führen Sie Ihren Test in einer benutzerdefinierten Umgebung aus.
6. Wählen Sie im Dropdownmenü mit den Testspezifikationen die Testspezifikationsdatei aus, die Sie derzeit für Tests auf dem Legacy-Testhost verwenden.
7. Kopieren Sie den Inhalt dieser Datei und fügen Sie ihn lokal in einen Texteditor ein, damit Sie später darauf zurückgreifen können.
8. Ändern Sie im Dropdownmenü mit den Testspezifikationen Ihre Testspezifikationsauswahl auf die neueste Standardtestspezifikationsdatei.
9. Wählen Sie Bearbeiten und Sie gelangen zur Bearbeitungsoberfläche für Testspezifikationen. Sie werden feststellen, dass sich die Testspezifikationsdatei in den ersten Zeilen bereits für den neuen Testhost angemeldet hat:

```
android_test_host: amazon_linux_2
```

- 10 Lesen Sie [hier](#) die Syntax für die Auswahl von Testhosts und die wichtigsten Unterschiede zwischen den Testhosts [hier](#).
- 11 Fügen Sie selektiv Befehle aus Ihrer lokal gespeicherten Testspezifikationsdatei aus Schritt 6 in die neue Standard-Testspezifikationsdatei ein und bearbeiten Sie sie. Wählen Sie dann Speichern unter, um die neue Spezifikationsdatei zu speichern. Sie können jetzt Testläufe auf dem Amazon Linux 2-Testhost planen.

Unterschiede zwischen den neuen und den alten Testhosts

Wenn Sie Ihre Testspezifikationsdatei bearbeiten, um den Amazon Linux 2-Testhost zu verwenden und Ihre Tests vom alten Testhost umzustellen, sollten Sie sich der folgenden wichtigen Umgebungsunterschiede bewusst sein:

- **Softwareversionen auswählen:** In vielen Fällen haben sich die Standard-Softwareversionen geändert. Wenn Sie Ihre Softwareversion also nicht zuvor explizit auf dem Legacy-Testhost ausgewählt haben, möchten Sie sie jetzt möglicherweise auf dem Amazon Linux 2-Testhost mit angeben [devicefarm-cli](#). In den allermeisten Anwendungsfällen empfehlen wir Kunden, die von ihnen verwendeten Softwareversionen explizit auszuwählen. Wenn Sie eine Softwareversion mit `auswählendevicfarm-cli`, haben Sie eine vorhersehbare und konsistente Erfahrung damit und erhalten zahlreiche Warnungen, wenn Device Farm plant, diese Version vom Testhost zu entfernen.

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

- **Verfügbare Softwareversionen:** Viele Versionen zuvor vorinstallierter Software wurden entfernt und viele neue Versionen hinzugefügt. Stellen Sie daher sicher, dass Sie bei der `devicefarm-cli` Auswahl Ihrer Softwareversionen Versionen auswählen, die in der [Liste der unterstützten Versionen enthalten](#) sind.
- **Alle Dateipfade,** die in Ihrer Legacy-Host-Testspezifikationsdatei als absolute Pfade fest codiert sind, funktionieren auf dem Amazon Linux 2-Testhost höchstwahrscheinlich nicht wie erwartet. Sie werden generell nicht für die Verwendung von Testspezifikationsdateien empfohlen. Wir empfehlen, relative Pfade und Umgebungsvariablen für den gesamten Code der Testspezifikationsdatei zu verwenden. Beachten Sie außerdem, dass die meisten Binärdateien, die Sie für Ihren Test benötigen, im PATH des Hosts zu finden sind, sodass sie sofort von der Spezifikationsdatei aus ausgeführt werden können, indem nur ihr Name verwendet wird (z. B. Appium).
- Die Erfassung von Leistungsdaten wird derzeit auf dem neuen Testhost nicht unterstützt.
- **Betriebssystemversion:** Der alte Testhost basierte auf dem Ubuntu-Betriebssystem, wohingegen der neue auf Amazon Linux 2 basiert. Infolgedessen stellen Benutzer möglicherweise einige Unterschiede zwischen den verfügbaren Systembibliotheken und den Versionen der Systembibliotheken fest.
- Für Appium Java-Benutzer enthält der neue Testhost keine vorinstallierten JAR-Dateien in seinem Klassenpfad, wohingegen der vorherige Host eine für das TestNG-Framework (über eine Umgebungsvariable) enthielt. `$DEVICEFARM_TESTNG_JAR` Wir empfehlen Kunden, die erforderlichen JAR-Dateien für ihre Testframeworks in ihr Testpaket zu packen und Instanzen der `$DEVICEFARM_TESTNG_JAR` Variablen aus ihren Testspezifikationsdateien zu entfernen. Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).
- Für Appium-Benutzer wurde die `$DEVICEFARM_CHROMEDRIVER_EXECUTABLE` Umgebungsvariable zugunsten eines neuen Ansatzes entfernt, mit dem Kunden auf Chromedriver für Android zugreifen können. Ein Beispiel, das eine neue Umgebungsvariable verwendet, finden Sie in unserer [Standard-Testspezifikationsdatei](#).
`$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR`

Note

Wir empfehlen dringend, den vorhandenen Appium-Serverbefehl aus der Standard-Testspezifikationsdatei unverändert zu lassen.

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

Umgebungsvariablen

Umgebungsvariablen repräsentieren Werte, die von Ihren automatisierten Tests verwendet werden. Sie können diese Umgebungsvariablen in Ihren YAML-Dateien und Ihrem Testcode verwenden. In einer benutzerdefinierten Testumgebung füllt Device Farm Umgebungsvariablen zur Laufzeit dynamisch aus.

Themen

- [Allgemeine Umgebungsvariablen](#)
- [Appium-Java-JUnit-Umgebungsvariablen](#)
- [Appium-Java-TestNG-Umgebungsvariablen](#)
- [XCUI-Test-Umgebungsvariablen](#)

Allgemeine Umgebungsvariablen

Android-Tests

In diesem Abschnitt werden benutzerdefinierte Umgebungsvariablen beschrieben, die Android-Plattformtests gemeinsam sind, die von Device Farm unterstützt werden.

\$DEVICEFARM_DEVICE_NAME

Name des Geräts, auf dem die Tests ausgeführt werden. Stellt die eindeutige Geräte-Kennung (UDID) des Geräts dar.

\$DEVICEFARM_DEVICE_PLATFORM_NAME

Der Name der Geräteplattform. Ist entweder Android oder iOS.

\$DEVICEFARM_DEVICE_OS_VERSION

Die Betriebssystemversion des Geräts.

\$DEVICEFARM_APP_PATH

Der Pfad zur mobilen App auf dem Host-Computer, auf dem die Tests ausgeführt werden. Der App-Pfad ist nur für mobile Apps verfügbar.

\$DEVICEFARM_DEVICE_UDID

Die eindeutige Kennung des mobilen Geräts, auf dem automatisierte Tests ausgeführt werden.

\$DEVICEFARM_LOG_DIR

Der Pfad zu den Protokolldateien, die während des Testlaufs generiert werden. Standardmäßig werden alle Dateien in diesem Verzeichnis in einer ZIP-Datei archiviert und nach Ihrem Testlauf als Artefakt zur Verfügung gestellt.

\$DEVICEFARM_SCREENSHOT_PATH

Der Pfad zu den Screenshots, die ggf. während des Testlaufs erfasst werden.

\$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR

Der Speicherort eines Verzeichnisses, das die erforderlichen ausführbaren Chromedriver-Dateien für die Verwendung in Appium-Web- und Hybridtests enthält.

\$ANDROID_HOME

Der Pfad zum Android SDK-Installationsverzeichnis.

Note

Die ANDROID_HOME Umgebungsvariable ist nur auf dem Amazon Linux 2-Testhost für Android verfügbar.

iOS-Tests

In diesem Abschnitt werden benutzerdefinierte Umgebungsvariablen beschrieben, die den von Device Farm unterstützten iOS-Plattformtests gemeinsam sind.

\$DEVICEFARM_DEVICE_NAME

Name des Geräts, auf dem die Tests ausgeführt werden. Stellt die eindeutige Geräte-Kennung (UDID) des Geräts dar.

\$DEVICEFARM_DEVICE_PLATFORM_NAME

Der Name der Geräteplattform. Ist entweder Android oder iOS.

\$DEVICEFARM_APP_PATH

Der Pfad zur mobilen App auf dem Host-Computer, auf dem die Tests ausgeführt werden. Der App-Pfad ist nur für mobile Apps verfügbar.

\$DEVICEFARM_DEVICE_UDID

Die eindeutige Kennung des mobilen Geräts, auf dem automatisierte Tests ausgeführt werden.

\$DEVICEFARM_LOG_DIR

Der Pfad zu den Protokolldateien, die während des Testlaufs generiert werden.

\$DEVICEFARM_SCREENSHOT_PATH

Der Pfad zu den Screenshots, die ggf. während des Testlaufs erfasst werden.

Appium-Java-JUnit-Umgebungsvariablen

In diesem Abschnitt werden Umgebungsvariablen beschrieben, die von Appium Java JUnit-Tests in einer benutzerdefinierten Testumgebung verwendet werden.

\$DEVICEFARM_TESTNG_JAR

Der Pfad zur TestNG .jar-Datei.

\$DEVICEFARM_TEST_PACKAGE_PATH

Der Pfad zum entpackten Inhalt der Test-Paketdatei.

Appium-Java-TestNG-Umgebungsvariablen

In diesem Abschnitt werden Umgebungsvariablen beschrieben, die von Appium Java TestNG-Tests in einer benutzerdefinierten Testumgebung verwendet werden.

\$DEVICEFARM_TESTNG_JAR

Der Pfad zur TestNG .jar-Datei.

\$DEVICEFARM_TEST_PACKAGE_PATH

Der Pfad zum entpackten Inhalt der Test-Paketdatei.

XCUITest-Umgebungsvariablen

\$DEVICEFARM_XCUITESTRUN_FILE

Pfad zur Device Farm- .xcctestunDatei. Wird über Ihre App und aus Testpaketen erstellt.

\$DEVICEFARM_DERIVED_DATA_PATH

Erwarteter Pfad der xcodebuild-Ausgabe von Device Farm.

Migrieren von Tests von einer Standard-Testumgebung zu einer benutzerdefinierten Testumgebung

Im folgenden Handbuch wird erläutert, wie Sie von einem Standard-Testausführungsmodus zu einem benutzerdefinierten Ausführungsmodus wechseln. Die Migration umfasst hauptsächlich zwei verschiedene Ausführungsformen:

1. **Standardmodus** : Dieser Testausführungsmodus wurde hauptsächlich entwickelt, um Kunden detaillierte Berichte und eine vollständig verwaltete Umgebung zu bieten.
2. **Benutzerdefinierter Modus**: Dieser Testausführungsmodus wurde für verschiedene Anwendungsfälle entwickelt, die schnellere Testläufe erfordern, die Fähigkeit, sich zu heben und zu verschieben und Parität mit ihrer lokalen Umgebung zu erreichen, und Live-Videostreaming.

Überlegungen bei der Migration

In diesem Abschnitt werden einige der wichtigsten Anwendungsfälle aufgeführt, die bei der Migration in den benutzerdefinierten Modus zu beachten sind:

1. **Geschwindigkeit** : Im Standardausführungsmodus analysiert Device Farm die Metadaten der Tests, die Sie mithilfe der Paketierungsanweisungen für Ihr spezielles Framework verpackt und hochgeladen haben. Beim Parsen wird die Anzahl der Tests in Ihrem Paket erkannt. Danach führt Device Farm jeden Test separat aus und zeigt die Protokolle, Videos und andere Ergebnisartefakte für jeden Test einzeln an. Dies erhöht jedoch stetig die gesamte end-to-end Testausführungszeit, da es die Vor- und Nachverarbeitung von Tests und Ergebnisartefakten am Service-Ende gibt.

Im Gegensatz dazu analysiert der benutzerdefinierte Ausführungsmodus Ihr Testpaket nicht. Dies bedeutet keine Vorverarbeitung und minimale Nachverarbeitung für Tests oder Ergebnisartefakte. Dies führt zu end-to-end Gesamtausführungszeiten in der Nähe Ihrer lokalen Einrichtung. Die Tests werden im gleichen Format ausgeführt wie bei Ausführung auf Ihrem/Ihren lokalen Computer(n). Die Ergebnisse der Tests entsprechen denen, die Sie lokal erhalten, und stehen am Ende der Auftragsausführung zum Download zur Verfügung.

2. Anpassung oder Flexibilität: Der Standardausführungsmodus analysiert Ihr Testpaket, um die Anzahl der Tests zu erkennen, und führt dann jeden Test separat aus. Beachten Sie, dass es keine Garantie dafür gibt, dass die Tests in der von Ihnen angegebenen Reihenfolge ausgeführt werden. Daher funktionieren Tests, die eine bestimmte Ausführungssequenz erfordern, möglicherweise nicht wie erwartet. Darüber hinaus gibt es keine Möglichkeit, die Umgebung des Hostcomputers anzupassen oder Konfigurationsdateien zu übergeben, die möglicherweise erforderlich sind, um Ihre Tests auf bestimmte Weise auszuführen.

Im Gegensatz dazu können Sie mit dem benutzerdefinierten Modus die Host-Computerumgebung konfigurieren, einschließlich der Möglichkeit, zusätzliche Software zu installieren, Filter an Ihre Tests zu übergeben, Konfigurationsdateien zu übergeben und die Einrichtung der Testausführung zu steuern. Dies wird über eine Yaml-Datei (auch Testspezifikationsdatei genannt) erreicht, die Sie ändern können, indem Sie Shell-Befehle hinzufügen. Diese Yaml-Datei wird in ein Shell-Skript konvertiert, das auf dem Testhost-Computer ausgeführt wird. Sie können mehrere yaml-Dateien speichern und eine dynamisch nach Ihren Anforderungen auswählen, wenn Sie eine Ausführung planen.

3. Live-Video und Protokollierung: Sowohl Standard- als auch benutzerdefinierte Ausführungsmodi bieten Ihnen Videos und Protokolle für Ihre Tests. Im Standardmodus erhalten Sie jedoch die Video- und vordefinierten Protokolle Ihrer Tests erst, nachdem Ihre Tests abgeschlossen sind.

Im Gegensatz dazu bietet Ihnen der benutzerdefinierte Modus einen Livestream der Video- und clientseitigen Protokolle Ihrer Tests. Darüber hinaus können Sie das Video und andere Artefakte am Ende des/der Test(e) herunterladen.

4. Veralterung: Die folgenden Testtypen werden bis Ende Dezember 2023 im Standardausführungsmodus veraltet sein:

- Appium (alle Sprachen)
- Calabash
- XCTest
- UI-Automatisierung

- UI Automator
- Webtests
- Integrierter Explorer

Nach der Veraltung können Sie diese Frameworks nicht mehr im Standardmodus verwenden. Sie können stattdessen den benutzerdefinierten Modus für die oben aufgeführten Testtypen verwenden.

Tip

Wenn Ihr Anwendungsfall mindestens einen der oben genannten Faktoren beinhaltet, empfehlen wir dringend, zum benutzerdefinierten Ausführungsmodus zu wechseln.

Schritte zur Migration

Gehen Sie wie folgt vor, um vom Standardmodus in den benutzerdefinierten Modus zu migrieren:

1. Melden Sie sich bei der an AWS Management Console und öffnen Sie die Device Farm-Konsole unter <https://console.aws.amazon.com/devicefarm/>.
2. Wählen Sie Ihr Projekt aus und starten Sie dann einen neuen Automatisierungslauf.
3. Laden Sie Ihre App hoch (oder wählen Sie `awsweb` app), wählen Sie Ihren Test-Framework-Typ, laden Sie Ihr Testpaket hoch und wählen Sie dann unter dem `Choose your execution environment` Parameter die Option zu `ausRun your test in a custom environment`.
4. Standardmäßig wird die Beispiel-Testspezifikationsdatei von Device Farm angezeigt, die Sie anzeigen und bearbeiten können. Diese Beispieldatei kann als Ausgangspunkt verwendet werden, um Ihre Tests im [benutzerdefinierten Umgebungsmodus](#) auszuprobieren. Sobald Sie dann von der Konsole aus überprüft haben, dass Ihre Tests ordnungsgemäß funktionieren, können Sie jede Ihrer API-, CLI- und Pipeline-Integrationen mit Device Farm so ändern, dass diese Testspezifikationsdatei als Parameter bei der Planung von Testläufen verwendet wird. Informationen zum Hinzufügen einer Testspezifikationsdatei als Parameter für Ihre Läufe finden Sie im `testSpecArn` Parameterabschnitt für die `-ScheduleRunAPI` in unserem [API-Leitfaden](#).

Appium-Framework

In einer benutzerdefinierten Testumgebung fügt Device Farm keine Appium-Funktionen in Ihre Appium-Framework-Tests ein oder überschreibt sie. Sie müssen die Appium-Funktionen Ihres Tests entweder in der YAML-Datei der Testspezifikation oder im Testcode angeben.

Android-Instrumentierung

Für die Migration Ihrer Android-Instrumentierungstests auf eine benutzerdefinierte Testumgebung müssen Sie keine Änderungen vornehmen.

iOS XCUITest

Für die Migration Ihrer iOS XCUITest-Tests auf eine benutzerdefinierte Testumgebung müssen Sie keine Änderungen vornehmen.

Erweiterung benutzerdefinierter Testumgebungen in Device Farm

Im benutzerdefinierten Modus von Device Farm können Sie mehr als nur Ihre Testsuite ausführen. In diesem Abschnitt erfahren Sie, wie Sie Ihre Testsuite erweitern und Ihre Tests optimieren können.

Eine PIN festlegen

Bei einigen Anwendungen müssen Sie eine PIN auf dem Gerät festlegen. Device Farm unterstützt das native Einstellen einer PIN auf Geräten nicht. Dies ist jedoch mit den folgenden Einschränkungen möglich:

- Auf dem Gerät muss Android 8 oder höher ausgeführt werden.
- Die PIN muss nach Abschluss des Tests entfernt werden.

Um die PIN in Ihren Tests festzulegen, verwenden Sie die `post_test` Phasen `pre_test` und, um die PIN festzulegen und zu entfernen, wie im Folgenden gezeigt:

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

- ```
- # ... Among your post_test commands
- adb shell locksettings clear --old "$DEVICE_PIN_CODE"
```

Wenn Ihre Testsuite beginnt, ist die PIN 1234 festgelegt. Nach dem Beenden Ihrer Testsuite wird die PIN entfernt.

### Warning

Wenn Sie die PIN nach Abschluss des Tests nicht vom Gerät entfernen, werden das Gerät und Ihr Konto unter Quarantäne gestellt.

## Beschleunigung von APPIUM-basierten Tests durch die gewünschten Funktionen

Wenn Sie Appium verwenden, stellen Sie möglicherweise fest, dass die Testsuite im Standardmodus sehr langsam ist. Dies liegt daran, dass Device Farm die Standardeinstellungen anwendet und keine Annahmen darüber trifft, wie Sie die Appium-Umgebung verwenden möchten. Diese Standardeinstellungen basieren zwar auf den bewährten Methoden der Branche, gelten jedoch möglicherweise nicht für Ihre Situation. Um die Parameter des Appium-Servers zu optimieren, können Sie die Standardfunktionen von Appium in Ihrer Testspezifikation anpassen. Im Folgenden wird beispielsweise die `usePrebuiltWDA` Fähigkeit `true` in einer iOS-Testsuite auf festgelegt, um die anfängliche Startzeit zu beschleunigen:

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

Bei Appium-Fähigkeiten muss es sich um eine JSON-Struktur mit Shell-Escape-Code in Anführungszeichen handeln.

Die folgenden Appium-Funktionen sind häufig Quellen für Leistungsverbesserungen:

#### `noReset` und `fullReset`

Diese beiden Funktionen, die sich gegenseitig ausschließen, beschreiben das Verhalten von Appium nach Abschluss jeder Sitzung. Wenn auf gesetzt `noReset` ist `true`, entfernt der Appium-Server keine Daten aus Ihrer Anwendung, wenn eine Appium-Sitzung endet, und führt praktisch keine Bereinigung durch. `fullReset` deinstalliert und löscht alle Anwendungsdaten vom Gerät, nachdem die Sitzung geschlossen wurde. Weitere Informationen finden Sie unter [Reset-Strategien](#) in der Appium-Dokumentation.

#### `ignoreUnimportantViews`(Nur Android)

Weist Appium an, die Hierarchie der Android-Benutzeroberfläche nur auf die für den Test relevanten Ansichten zu komprimieren, wodurch die Suche nach bestimmten Elementen beschleunigt wird. Dies kann jedoch einige XPath-basierte Testsuiten beschädigen, da die Hierarchie des UI-Layouts geändert wurde.

#### `skipUnlock`(Nur Android)

Informiert Appium darüber, dass derzeit kein PIN-Code festgelegt ist, was Tests nach einem Ausschalten des Bildschirms oder einem anderen Sperrenereignis beschleunigt.

#### `webDriverAgentUrl`(nur iOS)

Weist Appium an, davon auszugehen, dass eine wichtige iOS-Abhängigkeit, `webDriverAgent`, bereits läuft und für die Annahme von HTTP-Anfragen an der angegebenen URL verfügbar ist. Wenn Appium noch `webDriverAgent` nicht aktiv ist, kann es zu Beginn einer Testsuite einige Zeit dauern, bis Appium gestartet ist. `webDriverAgent` Wenn Sie `webDriverAgent` selbst starten und `http://localhost:8100` beim Start von Appium `webDriverAgentUrl` auf einstellen, können Sie Ihre Testsuite schneller starten. Beachten Sie, dass diese Funktion niemals zusammen mit der `useNewWDA` Funktion verwendet werden sollte.

Sie können den folgenden Code verwenden, um mit Ihrer Testspezifikationsdatei am lokalen Port 8100 des Geräts zu beginnen `webDriverAgent` und sie dann an den lokalen Port des Testhosts weiterzuleiten 8100 (auf diese Weise können Sie den Wert auf setzen `webDriverAgentUrl` `http://localhost:8100`). Dieser Code sollte während der Installationsphase ausgeführt werden, nachdem der Code für die Einrichtung des Appium und der `webDriverAgent` Umgebungsvariablen definiert wurde:

```

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

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

```

Anschließend können Sie Ihrer Testspezifikationsdatei den folgenden Code hinzufügen, um sicherzustellen, dass sie erfolgreich `webDriverAgent` gestartet wurde. Dieser Code sollte am Ende der Vortestphase ausgeführt werden, nachdem sichergestellt wurde, dass Appium erfolgreich gestartet wurde:

```

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

```

Weitere Informationen zu den Funktionen, die Appium unterstützt, finden Sie unter Appium [Desired Capabilities in der Appium-Dokumentation](#).

## Verwenden Sie Webhooks und andere APIs nach der Ausführung Ihrer Tests

Sie können Device Farm einen Webhook aufrufen lassen, nachdem die Verwendung curl jeder Testsuite abgeschlossen ist. Der dazu erforderliche Vorgang hängt vom Ziel und der Formatierung ab. Informationen zu Ihrem spezifischen Webhook finden Sie in der Dokumentation zu diesem Webhook. Im folgenden Beispiel wird jedes Mal, wenn eine Testsuite fertig ist, eine Nachricht an einen Slack-Webhook gesendet:

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

Weitere Informationen zur Verwendung von Webhooks mit Slack findest du unter [Deine erste Slack-Nachricht mit Webhook versenden in der Slack-API-Referenz](#).

Du bist nicht darauf beschränkt, Webhooks aufzurufen. curl Testpakete können zusätzliche Skripts und Tools enthalten, sofern sie mit der Device Farm Farm-Ausführungsumgebung kompatibel sind. Ihr Testpaket kann beispielsweise Hilfsskripten enthalten, die Anfragen an andere APIs stellen. Stellen Sie sicher, dass alle erforderlichen Pakete zusammen mit den Anforderungen Ihrer Testsuite installiert sind. Um ein Skript hinzuzufügen, das ausgeführt wird, nachdem Ihre Testsuite fertiggestellt ist, nehmen Sie das Skript in Ihr Testpaket auf und fügen Sie Ihrer Testspezifikation Folgendes hinzu:

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

### Note

Die Verwaltung aller API-Schlüssel oder anderer Authentifizierungstoken, die in Ihrem Testpaket verwendet werden, liegt in Ihrer Verantwortung. Wir empfehlen Ihnen, jegliche Form von Sicherheitsanmeldedaten außerhalb der Quellcodeverwaltung aufzubewahren, Anmeldeinformationen mit möglichst wenigen Rechten zu verwenden und wann immer möglich widerrufbare, kurzlebige Token zu verwenden. Informationen zur Überprüfung der

Sicherheitsanforderungen finden Sie in der Dokumentation der von Ihnen verwendeten Drittanbieter-APIs.

Wenn Sie planen, AWS Dienste als Teil Ihrer Testausführungssuite zu verwenden, sollten Sie temporäre IAM-Anmeldeinformationen verwenden, die außerhalb Ihrer Testsuite generiert und in Ihrem Testpaket enthalten sind. Für diese Anmeldeinformationen sollten die wenigsten erteilten Berechtigungen und die kürzestmögliche Lebensdauer gelten. Weitere Informationen zum Erstellen temporärer Anmeldeinformationen finden Sie unter [Temporäre Sicherheitsanmeldeinformationen anfordern](#) im IAM-Benutzerhandbuch.

## Zusätzliche Dateien zu Ihrem Testpaket hinzufügen

Möglicherweise möchten Sie zusätzliche Dateien als Teil Ihrer Tests verwenden, entweder als zusätzliche Konfigurationsdateien oder als zusätzliche Testdaten. Sie können diese zusätzlichen Dateien Ihrem Testpaket hinzufügen, bevor Sie es hochladen AWS Device Farm, und dann im benutzerdefinierten Umgebungsmodus darauf zugreifen. Grundsätzlich handelt es sich bei allen Uploadformaten für Testpakete (ZIP, IPA, APK, JAR usw.) um Paketarchivformate, die standardmäßige ZIP-Operationen unterstützen.

Sie können Ihrem Testarchiv Dateien hinzufügen, bevor Sie es hochladen, AWS Device Farm indem Sie den folgenden Befehl verwenden:

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

Für ein Verzeichnis mit zusätzlichen Dateien:

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

Diese Befehle funktionieren erwartungsgemäß für alle Upload-Formate von Testpaketen mit Ausnahme von IPA-Dateien. Für IPA-Dateien, insbesondere wenn sie mit XCUI Tests verwendet werden, empfehlen wir, dass Sie alle zusätzlichen Dateien aufgrund der Art und Weise, wie AWS Device Farm iOS-Testpakete entworfen werden, an einem etwas anderen Speicherort ablegen. Beim Erstellen Ihres iOS-Tests befindet sich das Testanwendungsverzeichnis in einem anderen Verzeichnis namens *PayLoad*.

So könnte beispielsweise ein solches iOS-Testverzeichnis aussehen:

```
$ tree
```

```

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

```

*Fügen Sie für diese XCUI-Test-Pakete dem Verzeichnis, das auf `.app` endet, alle zusätzlichen Dateien innerhalb des Payload-Verzeichnisses hinzu.* Die folgenden Befehle zeigen beispielsweise, wie Sie diesem Testpaket eine Datei hinzufügen können:

```
$ mv extra_file Payload/*.app/
```

```
$ zip -r my_xcui_tests.ipa Payload/
```

Wenn Sie Ihrem Testpaket eine Datei hinzufügen, können Sie AWS Device Farm je nach Upload-Format ein leicht unterschiedliches Interaktionsverhalten erwarten. Wenn für den Upload die ZIP-Dateierweiterung verwendet wurde, AWS Device Farm wird der Upload vor dem Test automatisch entpackt und die entpackten Dateien werden an dem Speicherort mit der Umgebungsvariablen `$DEVICEFARM_TEST_PACKAGE_PATH` belassen. *(Das heißt, wenn Sie wie im ersten Beispiel eine Datei namens `extra_file` zum Stammverzeichnis des Archivs hinzufügen würden, würde sie sich während des Tests unter `$DEVICEFARM_Test_Package_Path/extra_file` befinden).*

Um ein praktischeres Beispiel zu verwenden: Wenn Sie ein Appium TestNG-Benutzer sind und Ihrem Test eine Datei `testng.xml` hinzufügen möchten, können Sie sie mit dem folgenden Befehl in Ihr Archiv aufnehmen:

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

Anschließend können Sie Ihren Testbefehl im benutzerdefinierten Umgebungsmodus wie folgt ändern:

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

Wenn Ihre Upload-Erweiterung für das Testpaket nicht ZIP ist (z. B. APK-, IPA- oder JAR-Datei), befindet sich die hochgeladene Paketdatei selbst unter `$DEVICEFARM_TEST_PACKAGE_PATH`. Da es sich immer noch um Dateien im Archivformat handelt, können Sie die Datei entpacken, um von dort aus auf die zusätzlichen Dateien zuzugreifen. *Mit dem folgenden Befehl wird beispielsweise der Inhalt des Testpakets (für APK-, IPA- oder JAR-Dateien) in das Verzeichnis `/tmp` entpackt:*

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

*Im Fall einer APK- oder JAR-Datei würden Sie feststellen, dass Ihre zusätzlichen Dateien in das Verzeichnis `/tmp` entpackt wurden (z. B. `/tmp/extra_file`). Im Fall einer IPA-Datei würden sich zusätzliche Dateien, wie bereits erläutert, an einem etwas anderen Ort innerhalb des Ordners befinden, der auf `.app` endet und sich innerhalb des Payload-Verzeichnisses*

*befindet*. Basierend auf dem obigen IPA-Beispiel würde sich die Datei beispielsweise im Verzeichnis */tmp/payload/ADFIOS ReferenceApp UITests-Runner.app/extra\_file* befinden (referenzierbar als */tmp/Payload/ \*.app/extra\_file*).

# Arbeiten mit Fernzugriff in AWS Device Farm

Der Remotezugriff ermöglicht, Geräte über Ihren Webbrowser in Echtzeit zu bedienen, inklusive Fingergesten, Gesten und anderen Interaktionen, beispielsweise, um Funktionalitäten zu testen und Kundenprobleme zu reproduzieren. Sie interagieren mit einem bestimmten Gerät, indem Sie ein Remotezugriffssitzung mit diesem Gerät erstellen.

Eine Sitzung in Device Farm ist eine Interaktion in Echtzeit mit einem tatsächlichen, physischen Gerät, das in einem Webbrowser gehostet wird. In einer Sitzung wird das einzelne Gerät angezeigt, das Sie beim Start der Sitzung ausgewählt haben. Ein Benutzer kann mehrere Sitzungen gleichzeitig starten, wobei die Gesamtanzahl von gleichzeitigen Geräten durch die Anzahl der Geräteplätze, die Sie zur Verfügung haben, beschränkt wird. Sie können Geräteplätze für bestimmte Gerätefamilien (Android- oder iOS-Geräte) erwerben. Weitere Informationen finden Sie unter [Device Farm – Preise](#).

Device Farm bietet derzeit eine Untergruppe von Geräten für Fernzugriffstests. Diesem Gerätepool werden laufend neue Geräte hinzugefügt.

Device Farm zeichnet Videos von jeder Fernzugriffssitzung auf und generiert Aktivitätsprotokolle während der Sitzung. Diese Ergebnisse enthalten alle Informationen, die Sie während einer Sitzung eingeben.

## Note

Aus Sicherheitsgründen empfehlen wir Ihnen, in einer Remotezugriffssitzung die Angabe oder Eingabe von sensiblen Daten, wie beispielsweise Kontonummern, persönlichen Anmeldeinformationen und anderen Details, zu vermeiden.

## Themen

- [Erstellen Sie eine Fernzugriffssitzung in AWS Device Farm](#)
- [Verwenden Sie eine Fernzugriffssitzung in AWS Device Farm](#)
- [Ergebnisse einer Fernzugriffssitzung in AWS Device Farm abrufen](#)

## Erstellen Sie eine Fernzugriffssitzung in AWS Device Farm

Weitere Informationen zu Remotezugriffssitzungen finden Sie unter [Sitzungen](#).

- [Voraussetzungen](#)
- [Erstellen Sie einen Testlauf \(Konsole\)](#)
- [Nächste Schritte](#)

## Voraussetzungen

- Erstellen Sie ein Projekt in Device Farm. Befolgen Sie die Anweisungen unter [Erstellen Sie ein Projekt in AWS Device Farm](#), und kehren Sie dann zu dieser Seite zurück.

## Erstellen Sie eine Sitzung mit der Device Farm-Konsole

1. Melden Sie sich bei der Device Farm-Konsole an unter <https://console.aws.amazon.com/devicefarm>.
2. Wählen Sie im Device Farm-Navigationsbereich Testen von Mobilgeräten und wählen Sie dann Projekte.
3. Wenn Sie bereits ein Projekt haben, wählen Sie es aus der Liste aus. Andernfalls erstellen Sie ein Projekt, indem Sie den Anweisungen unter folgen [Erstellen Sie ein Projekt in AWS Device Farm](#).
4. Wählen Sie auf der Registerkarte Remote access (Remotezugriff) die Option Start a new session (Eine neue Sitzung starten).
5. Wählen Sie ein Gerät für Ihre Sitzung. Sie können aus der Liste der verfügbaren Geräte auswählen oder mithilfe der Suchleiste oben in der Liste nach einem Gerät suchen. Sie können nach folgenden Kriterien suchen:
  - Name
  - Plattform
  - Formfaktor
  - Art der Flotte
6. Geben Sie unter Session name (Sitzungsname) einen Namen für die Sitzung ein.
7. Wählen Sie Confirm and start session (Bestätigen und Sitzung starten).

## Nächste Schritte

Device Farm startet die Sitzung, sobald das angeforderte Gerät verfügbar ist, normalerweise innerhalb weniger Minuten. Das Dialogfeld wird angezeigt, bis die Sitzung beginnt. Um die Sitzungsanfrage abzubrechen, wählen Sie `Cancel request` (Anforderung abbrechen).

Wenn Sie nach dem Start einer Sitzung den Browser oder die Browser-Registerkarte schließen, ohne die Sitzung anzuhalten, oder wenn die Verbindung zwischen dem Browser und dem Internet verloren geht, bleibt die Sitzung für fünf weitere Minuten aktiv. Danach beendet Device Farm die Sitzung. Ihr Konto wird jedoch für die Leerlaufzeit belastet.

Nach dem Start der Sitzung können Sie über den Webbrowser mit dem Gerät interagieren.

## Verwenden Sie eine Fernzugriffssitzung in AWS Device Farm

Weitere Informationen zur Ausführung von interaktiven Tests mit Android- und iOS-Apps über Remotezugriffssitzungen finden Sie unter [Sitzungen](#).

- [Voraussetzungen](#)
- [Verwenden Sie eine Sitzung in der Device Farm-Konsole](#)
- [Nächste Schritte](#)
- [Tipps und Tricks](#)

## Voraussetzungen

- Erstellen Sie eine Sitzung. Befolgen Sie die Anweisungen unter [Erstellen Sie eine Sitzung](#), und kehren Sie dann zu dieser Seite zurück.

## Verwenden Sie eine Sitzung in der Device Farm-Konsole

Sobald das von Ihnen für eine Remotezugriffssitzung angeforderte Gerät verfügbar ist, zeigt die Konsole den Gerätebildschirm an. Die Sitzung dauert maximal 150 Minuten. Die verbleibende Zeit der Sitzung wird im `Verbleibende Zeit`-Feld neben dem Gerätenamen.

## Eine Anwendung installieren

Um eine Anwendung auf dem Sitzungsgerät zu installieren, wählen Sie `Installieren Sie Anwendungen`, wählen Sie die Datei und wählen Sie dann die APK-Datei (Android) oder die IPA-Datei (iOS).

aus, die Sie installieren möchten. Anwendungen, die Sie in einer Remotezugriffssitzung ausführen, erfordern keine Testinstrumentierung oder -bereitstellung.

### Note

AWS Device Farm zeigt nach der Installation einer App keine Bestätigung an. Versuchen Sie, mit dem App-Symbol zu interagieren, um festzustellen, ob die App verwendet werden kann. Wenn Sie eine App hochladen, gibt es manchmal eine Verzögerung, bevor die App verfügbar ist. Auf der Taskleiste können Sie erkennen, ob die App verfügbar ist.

## Das Gerät steuern

Sie können mit dem in der Konsole angezeigten Gerät wie mit dem echten physischen Gerät interagieren, indem Sie Ihre Maus oder ein ähnliches Gerät für die Touch-Eingabe und die Bildschirmtastatur des Geräts verwenden. Im Fall von Android-Geräten gibt es Schaltflächen in View controls (Steuerelemente anzeigen), die genau wie die Schaltflächen Start und Zurück auf einem Android-Gerät funktionieren. Im Fall von iOS-Geräten gibt es eine Schaltfläche namens Home, die genauso wie die Startschaltfläche auf einem iOS-Gerät funktioniert. Sie können auch zwischen Anwendungen wechseln, die auf dem Gerät ausgeführt werden, indem Sie Aktuelle Apps.

## Umschalten zwischen Hoch- und Querformat

Sie können für die Geräte, die Sie verwenden, auch zwischen dem Hochformat- (vertikal) und dem Querformat (horizontal) wechseln.

## Nächste Schritte

Device Farm setzt die Sitzung fort, bis Sie sie manuell beenden oder das Zeitlimit von 150 Minuten erreicht ist. Um die Sitzung zu beenden, wählen Sie Sitzung beenden. Nachdem die Sitzung beendet wurde, können Sie auf das erfasste Video und die erstellten Protokolle zugreifen. Weitere Informationen finden Sie unter [Sitzungsergebnisse abrufen](#).

## Tipps und Tricks

In einigen AWS-Regionen treten möglicherweise Leistungsprobleme mit der Remotezugriffssitzung auf. Dies liegt teilweise an der Latenz in einigen Regionen. Wenn Leistungsprobleme auftreten, lassen Sie der Remotesitzung etwas Zeit, um aufzuholen, bevor Sie erneut mit der App interagieren.

# Ergebnisse einer Fernzugriffssitzung in AWS Device Farm abrufen

Weitere Informationen zu Sitzungen finden Sie unter [Sitzungen](#).

- [Voraussetzungen](#)
- [Sitzungsdetails anzeigen](#)
- [Sitzungsvideos oder Sitzungsprotokolle werden heruntergeladen](#)

## Voraussetzungen

- Führen Sie eine Sitzung durch. Befolgen Sie die Anweisungen unter [Verwenden Sie eine Fernzugriffssitzung in AWS Device Farm](#), und kehren Sie dann zu dieser Seite zurück.

## Sitzungsdetails anzeigen

Wenn eine Fernzugriffssitzung endet, zeigt die Device Farm-Konsole eine Tabelle mit Details zu den Aktivitäten während der Sitzung an. Weitere Informationen finden Sie unter [Analysieren von Protokollinformationen](#).

Gehen Sie wie folgt vor, um die Details einer Sitzung zu einem späteren Zeitpunkt erneut anzuzeigen:

1. Wählen Sie im Device Farm-Navigationsbereich **Testen von Mobilgeräten** und wählen Sie dann **Projekte**.
2. Wählen Sie das Projekt aus, das die Sitzung enthält.
3. Wählen **Fernzugriff**, und wählen Sie dann die Sitzung, die Sie überprüfen möchten, aus der Liste aus.

## Sitzungsvideos oder Sitzungsprotokolle werden heruntergeladen

Wenn eine Fernzugriffssitzung endet, bietet die Device Farm-Konsole Zugriff auf eine Videoaufzeichnung der Sitzungs- und Aktivitätsprotokolle. Wählen Sie in den Sitzungsergebnissen die Registerkarte **Files (Dateien)**, um eine Liste mit Links zu den Sitzungsvideos und -protokollen anzuzeigen. Sie können diese Dateien im Browser anzeigen oder lokal speichern.

# Arbeiten mit privaten Geräten in AWS Device Farm

Ein privates Gerät ist ein physisches Mobilgerät, das AWS Device Farm in Ihrem Namen in einem Amazon-Rechenzentrum bereitstellt. Dieses Gerät ist exklusiv für Ihr AWS Konto verfügbar.

## Note

Derzeit sind private Geräte nur in der Region AWS USA West (Oregon) verfügbar (us-west-2).

Sobald Sie eine private Geräteflotte haben, können Sie Remote-Zugriffssitzungen erstellen oder unter Verwendung Ihrer privaten Geräte Testläufe planen. Sie können auch Instance-Profile erstellen, um das Verhalten von privaten Geräten während eines Testlaufs oder einer Remote-Zugriffssitzung zu steuern. Weitere Informationen finden Sie unter [Verwaltung privater Geräte in AWS Device Farm](#). Optional können Sie beantragen, dass bestimmte private Android-Geräte als gerootete Geräte bereitgestellt werden.

Sie können auch einen Amazon Virtual Private Cloud Cloud-Endpunktservice erstellen, um private Apps zu testen, auf die Ihr Unternehmen Zugriff hat, die aber nicht über das Internet erreichbar sind. Beispielsweise könnten Sie eine Webanwendung innerhalb Ihrer VPC ausführen, die Sie auf mobilen Geräten testen möchten. Weitere Informationen finden Sie unter [Verwendung von Amazon VPC-Endpunktservices mit Device Farm — Legacy \(nicht empfohlen\)](#).

Wenn Sie daran interessiert sind, eine Flotte von privaten Geräten zu verwenden, [kontaktieren Sie uns](#). Das Device Farm Team muss mit Ihnen zusammenarbeiten, um eine Flotte von privaten Geräten für Ihr AWS Konto einzurichten und bereitzustellen.

## Themen

- [Verwaltung privater Geräte in AWS Device Farm](#)
- [Private Geräte in einem Gerätepool auswählen](#)
- [Überspringen der erneuten Signierung von Apps auf privaten Geräten in AWS Device Farm](#)
- [Mit Amazon VPC zusammenarbeiten](#) [AWSRegionen](#)
- [Kündigung privater Geräte](#)

# Verwaltung privater Geräte in AWS Device Farm

Ein privates Gerät ist ein physisches Mobilgerät, das AWS Device Farm in Ihrem Namen in einem Amazon-Rechenzentrum bereitstellt. Dieses Gerät ist exklusiv für Ihr AWS-Konto verfügbar.

## Note

Derzeit sind private Geräte verfügbar in AWS Region USA West (Oregon) (us-west-2 nur).

Sie können eine Flotte mit einem oder mehreren privaten Geräten einrichten. Diese Geräte sind Ihrem AWS-Konto zugeordnet. Nachdem Sie die Geräte eingerichtet haben, können Sie optional ein oder mehrere Instance-Profile erstellen. Mithilfe der Instance-Profile werden die Testläufe automatisiert und auf den Gerät-Instances durchgängig dieselben Einstellungen angewendet.

In diesem Thema wird erläutert, wie Sie ein Instance-Profil erstellen und andere allgemeine Device-Management-Aufgaben ausführen.

## Themen

- [Erstellen eines Instance-Profils](#)
- [Verwaltung einer privaten Geräteinstanz](#)
- [Einen Testlauf erstellen oder eine Fernzugriffssitzung starten](#)
- [Nächste Schritte](#)

## Erstellen eines Instance-Profils

Um das Verhalten privater Geräte während eines Testlaufs oder einer Fernzugriffssitzung zu steuern, können Sie in Device Farm ein Instance-Profil erstellen oder ändern. Sie benötigen keine Instance-Profile, um mit der Verwendung Ihrer privaten Geräte beginnen zu können.

1. Öffnen Sie die Device Farm-Konsole unter <https://console.aws.amazon.com/devicefarm/>.
2. Wählen Sie im Device Farm-Navigationsbereich Testen von Mobilgeräten, dann wähle Private Geräte.
3. Wählen Sie Instance profiles (Instance-Profile) aus.
4. Wähle Instanzprofil erstellen.
5. Geben Sie einen Namen für das Instance-Profil ein.

## 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

- (Optional) Geben Sie eine Beschreibung für das Instance-Profil ein.
- (Optional) Ändern Sie eine der folgenden Einstellungen, um anzugeben, welche Aktionen Device Farm nach dem Ende jedes Testlaufs oder jeder Sitzung auf einem Gerät ausführen soll:
  - Nach der Verwendung neu starten— Um das Gerät neu zu starten, aktivieren Sie dieses Kontrollkästchen. Das Kontrollkästchen ist standardmäßig deaktiviert (`false`).
  - Bereinigung von Paketen— Um alle App-Pakete zu entfernen, die Sie auf dem Gerät installiert haben, aktivieren Sie dieses Kontrollkästchen. Das Kontrollkästchen ist standardmäßig deaktiviert (`false`). Wenn Sie alle App-Pakete, die Sie auf dem Gerät installiert haben, beibehalten möchten, lassen Sie das Kontrollkästchen deaktiviert.

- Pakete von der Bereinigung ausschließen— Um nur ausgewählte App-Pakete auf dem Gerät zu behalten, wählen Sie **Bereinigung von Paketen** aktivieren Sie das Kontrollkästchen, und wählen Sie dann **Neues hinzufügen**. Geben Sie als Paketnamen den vollständig qualifizierten Namen des App-Pakets ein, das Sie auf dem Gerät behalten möchten (z. B. `com.test.example`). Wählen Sie **Add new** (Neue hinzufügen) aus, wenn Sie mehrere App-Pakete auf dem Gerät beibehalten möchten. Geben Sie anschließend den vollständig qualifizierten Namen der einzelnen Pakete ein.
8. Wählen Sie **Speichern** aus.

## Verwaltung einer privaten Geräteinstanz

Wenn Sie bereits über eine oder mehrere private Geräte in Ihrer Flotte verfügen, können Sie sich für jedes Gerät Informationen anzeigen lassen und bestimmte Einstellungen verwalten. Sie können auch eine zusätzliche private Geräte-Instance anfordern.

1. Öffnen Sie die Device Farm-Konsole unter <https://console.aws.amazon.com/devicefarm/>.
2. Wählen Sie im Device Farm-Navigationsbereich **Testen von Mobilgeräten**, dann wähle **Private Geräte**.
3. Wählen Sie **Device Instances** (Geräte-Instances) aus. Die Registerkarte **Device instances** (Geräte-Instances) zeigt Ihnen eine Tabelle mit den privaten Geräten Ihrer Flotte an. Um die Tabelle schnell zu durchsuchen oder zu filtern, geben Sie Suchbegriffe in die Suchleiste über den Spalten ein.
4. (Optional) Um eine neue private Geräteinstanz anzufordern, wählen Sie **Geräteinstanz anfordern** oder [kontaktiere uns](#). Private Geräte erfordern eine zusätzliche Einrichtung mit Hilfe des Device Farm-Teams.
5. Wählen Sie in der Tabelle der Geräteinstanzen die Option zum Umschalten neben der Instanz aus, zu der Sie Informationen anzeigen oder die Sie verwalten möchten, und wählen Sie dann **Bearbeiten**.

### 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

- (Optional) Wählen Sie unter Profile (Profil) ein Instance-Profil aus, das dem Gerät angefügt werden soll. Dies kann hilfreich sein, wenn Sie beispielsweise eine bestimmte App immer von den Bereinigungsaufgaben ausschließen möchten.
- (Optional) Wählen Sie unter Labels (Bezeichnungen) die Option Add new (Neue hinzufügen) aus, um der Geräte-Instance eine neue Bezeichnung hinzuzufügen. Mithilfe von Bezeichnungen können Sie Geräte leichter kategorisieren und spezifische Geräte einfacher finden.
- Wählen Sie Speichern aus.

## Einen Testlauf erstellen oder eine Fernzugriffssitzung starten

Nachdem Sie eine private Geräteflotte eingerichtet haben, können Sie Testläufe erstellen oder Remote-Zugriffssitzungen mit einem oder mehreren privaten Geräten in Ihrer Flotte starten.

1. Öffnen Sie die Device Farm-Konsole unter <https://console.aws.amazon.com/devicefarm/>.
2. Wählen Sie im Device Farm-Navigationsbereich Testen von Mobilgeräten, dann wähle Projekte.
3. Wählen Sie ein vorhandenes Projekt aus der Liste aus oder erstellen Sie ein neues. Um ein neues Projekt zu erstellen, wählen Sie Neues Projekt, geben Sie einen Namen für das Projekt ein und wählen Sie dann Einreichen.
4. Führen Sie eine der folgenden Aktionen aus:
  - Wählen Sie zum Erstellen eines Testlaufs Automated tests (Automatisierte Tests) und anschließend Create a new run (Einen neuen Testlauf erstellen) aus. Der Assistent führt Sie durch die Schritte zum Erstellen des Testlaufs. Für die Geräte auswählen Schritt, Sie können einen vorhandenen Gerätepool bearbeiten oder einen neuen Gerätepool erstellen, der nur die privaten Geräte enthält, die das Device Farm-Team eingerichtet und mit Ihrem Gerät verknüpft hat AWS Konto. Weitere Informationen finden Sie unter [the section called “Erstellen Sie einen privaten Gerätepool”](#).
  - Wählen Sie für eine Remote-Zugriffssitzung Remote access (Remote-Zugriff) und dann Start a new Session (Starten einer neuen Sitzung) aus. Auf dem Wähle ein Gerät Seite, wählen Nur private Geräteinstanzen um die Liste nur auf die privaten Geräte zu beschränken, die das Device Farm-Team eingerichtet und mit Ihrem verknüpft hat AWS Konto. Wählen Sie anschließend die Geräte aus, auf die Sie während der Remote-Zugriffssitzung zugreifen möchten, und klicken Sie auf Confirm and start session (Bestätigen und Sitzung starten).

## 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 |
|-----------------------|-----------------------|-----------|----------|----|-------------|-------------|--------|
| <input type="radio"/> | OnePlus 8T            | AVAILABLE | Android  | 11 | Phone       | -           | -      |
| <input type="radio"/> | Samsung Galaxy Tab S7 | AVAILABLE | Android  | 11 | Tablet      | -           | -      |

## Nächste Schritte

Nach dem Einrichten Ihrer privaten Geräte stehen Ihnen für die Verwaltung Ihrer privaten Geräte folgende Möglichkeiten zur Verfügung:

- [Überspringen der erneuten Signatur von Apps auf privaten Geräten](#)
- [Verwenden Sie die Amazon Virtual Private Cloud-Endpunktdienste mit Device Farm](#)

Um ein Instanzprofil zu löschen, klicken Sie auf Instanzprofile. Wählen Sie im Menü die Umschaltoption neben der Instanz, die Sie löschen möchten, und wählen Sie dann Löschen.

## Private Geräte in einem Gerätepool auswählen

Um private Geräte in Ihrem Testlauf zu verwenden, können Sie einen Gerätepool erstellen, der Ihre privaten Geräte auswählt. Gerätepools ermöglichen es Ihnen, private Geräte hauptsächlich anhand von drei Arten von Gerätepoolregeln auszuwählen:

1. Regeln, die auf dem Geräte-ARN basieren
2. Regeln, die auf dem Geräteinstanzlabel basieren
3. Regeln, die auf dem ARN der Geräteinstanz basieren

In den folgenden Abschnitten werden die einzelnen Regeltypen und ihre Anwendungsfälle ausführlich beschrieben. Sie können die Device Farm-Konsole verwenden, AWS Befehlszeilenschnittstelle

(AWSCLI) oder die Device Farm API, um mithilfe dieser Regeln einen Gerätepool mit privaten Geräten zu erstellen oder zu ändern.

## Themen

- [Geräte-ARN](#)
- [Labels der Geräteinstanzen](#)
- [Instance-ARN](#)
- [Erstellen eines privaten Gerätepools mit privaten Geräten \(Konsole\)](#)
- [Einen privaten Gerätepool mit privaten Geräten erstellen \(AWS CLI\)](#)
- [Erstellen eines privaten Gerätepools mit privaten Geräten \(API\)](#)

## Geräte-ARN

Ein Geräte-ARN ist eine Kennung, die eher für einen Gerätetyp als für eine bestimmte physische Geräteinstanz steht. Ein Gerätetyp wird durch die folgenden Attribute definiert:

- Die Flotten-ID des Geräts
- Das Gerät ist OEM
- Die Modellnummer des Geräts
- Die Betriebssystemversion des Geräts
- Der Status des Geräts, der angibt, ob es gerootet ist oder nicht

Viele physische Geräteinstanzen können durch einen einzigen Gerätetyp dargestellt werden, wobei jede Instanz dieses Typs dieselben Werte für diese Attribute hat. Wenn Sie zum Beispiel drei hätten *Apple iPhone 13* Geräte auf der iOS-Version *16.1.0*. In Ihrer privaten Flotte würde sich jedes Gerät denselben Geräte-ARN teilen. Wenn Geräte mit denselben Attributen zu Ihrer Flotte hinzugefügt oder daraus entfernt würden, würde der Geräte-ARN weiterhin die verfügbaren Geräte in Ihrer Flotte für diesen Gerätetyp darstellen.

Der Geräte-ARN ist die robusteste Methode, private Geräte für einen Gerätepool auszuwählen, da er es dem Gerätepool ermöglicht, weiterhin Geräte auszuwählen, unabhängig von den spezifischen Geräteinstanzen, die Sie zu einem bestimmten Zeitpunkt bereitgestellt haben. Bei einzelnen privaten Geräteinstanzen kann es zu Hardwarefehlern kommen, sodass Device Farm sie automatisch durch neue funktionierende Instanzen desselben Gerätetyps ersetzt. In diesen Szenarien stellt die Geräte-ARN-Regel sicher, dass Ihr Gerätepool bei einem Hardwarefehler weiterhin Geräte auswählen kann.

Wenn Sie eine Geräte-ARN-Regel für private Geräte in Ihrem Gerätepool verwenden und einen Testlauf mit diesem Pool planen, überprüft Device Farm automatisch, welche privaten Geräteinstanzen durch diesen Geräte-ARN repräsentiert werden. Von den derzeit verfügbaren Instanzen wird eine davon für die Ausführung Ihres Tests zugewiesen. Wenn derzeit keine Instanzen verfügbar sind, wartet Device Farm, bis die erste verfügbare Instanz dieses Geräte-ARN verfügbar ist, und weist sie zur Ausführung Ihres Tests zu.

## Labels der Geräteinstanzen

Ein Geräteinstanzlabel ist eine Textkennung, die Sie als Metadaten für eine Geräteinstanz anhängen können. Sie können jeder Geräteinstanz mehrere Labels und mehreren Geräteinstanzen dasselbe Label zuordnen. Weitere Informationen zum Hinzufügen, Ändern oder Entfernen von Gerätelabels zu Geräteinstanzen finden Sie unter [Private Geräte verwalten](#).

Das Geräteinstanz-Label kann eine zuverlässige Methode zur Auswahl privater Geräte für einen Gerätepool sein, denn wenn Sie mehrere Geräteinstanzen mit derselben Bezeichnung haben, kann der Gerätepool eine beliebige davon für Ihren Test auswählen. Wenn der Geräte-ARN keine gute Regel für Ihren Anwendungsfall ist (wenn Sie beispielsweise aus Geräten mehrerer Gerätetypen auswählen möchten oder wenn Sie aus einer Teilmenge aller Geräte eines Gerätetyps auswählen möchten), können Sie mithilfe von Geräteinstanzbezeichnungen detaillierter aus mehreren Geräten für Ihren Gerätepool auswählen. Bei einzelnen privaten Geräteinstanzen kann es zu Hardwarefehlern kommen, sodass Device Farm sie automatisch durch neue funktionierende Instanzen desselben Gerätetyps ersetzt. In diesen Szenarien behält die Ersatzgeräteinstanz keine Metadaten zur Instanzbezeichnung des ersetzten Geräts bei. Wenn Sie also dieselbe Geräteinstanzbezeichnung auf mehrere Geräteinstanzen anwenden, stellt die Regel zur Geräteinstanzkennzeichnung sicher, dass Ihr Gerätepool bei einem Hardwarefehler weiterhin Geräteinstanzen auswählen kann.

Wenn Sie eine Geräteinstanz-Label-Regel für private Geräte in Ihrem Gerätepool verwenden und einen Testlauf mit diesem Pool planen, überprüft Device Farm automatisch, welche privaten Geräteinstanzen durch dieses Geräteinstanz-Label repräsentiert werden, und wählt aus diesen Instanzen nach dem Zufallsprinzip eine aus, die für die Ausführung Ihres Tests verfügbar ist. Wenn keine verfügbar sind, wählt Device Farm nach dem Zufallsprinzip eine Geräteinstanz mit dem Geräteinstanz-Label aus, um Ihren Test auszuführen, und stellt den Test in eine Warteschlange, damit er auf dem Gerät ausgeführt werden kann, sobald er verfügbar ist.

## Instance-ARN

Eine Geräteinstanz ARN ist eine Kennung, die eine physische Bare-Metal-Geräteinstanz darstellt, die in einer privaten Flotte eingesetzt wird. Wenn Sie zum Beispiel drei hätten *iPhone 13* Geräte auf

**OS15.0.0** In Ihrer privaten Flotte würde sich zwar jedes Gerät denselben Geräte-ARN teilen, aber jedes Gerät hätte auch seinen eigenen Instanz-ARN, der nur diese Instanz repräsentiert.

Der ARN der Geräteinstanz ist die am wenigsten robuste Methode zur Auswahl privater Geräte für einen Gerätepool und wird nur empfohlen, wenn die Geräte-ARNs und Geräteinstanzbezeichnungen nicht zu Ihrem Anwendungsfall passen. Geräteinstanz-ARNs werden häufig als Regeln für Gerätepools verwendet, wenn eine bestimmte Geräteinstanz als Voraussetzung für Ihren Test auf einzigartige und spezifische Weise konfiguriert ist und wenn diese Konfiguration bekannt und verifiziert sein muss, bevor der Test auf ihr ausgeführt wird. Bei einzelnen privaten Geräteinstanzen kann es zu Hardwarefehlern kommen, sodass Device Farm sie automatisch durch neue funktionierende Instanzen desselben Gerätetyps ersetzt. In diesen Szenarien hat die Ersatzgeräteinstanz einen anderen Geräteinstanz-ARN als das ersetzte Gerät. Wenn Sie sich also für Ihren Gerätepool auf Geräteinstanz-ARNs verlassen, müssen Sie die Regeldefinition Ihres Gerätepools manuell ändern und nicht mehr den alten ARN verwenden, sondern den neuen ARN verwenden. Wenn Sie das Gerät für den Test manuell vorkonfigurieren müssen, kann dies ein effektiver Arbeitsablauf sein (im Vergleich zu Geräte-ARNs). Für Tests in großem Maßstab wird empfohlen, diese Anwendungsfälle so anzupassen, dass sie mit Geräteinstanzbezeichnungen funktionieren, und wenn möglich, mehrere Geräteinstanzen für Tests vorkonfigurieren zu lassen.

Wenn Sie eine ARN-Regel für Geräteinstanzen für private Geräte in Ihrem Gerätepool verwenden und einen Testlauf mit diesem Pool planen, weist Device Farm diesen Test automatisch dieser Geräteinstanz zu. Wenn diese Geräteinstanz nicht verfügbar ist, stellt Device Farm den Test auf dem Gerät in eine Warteschlange, sobald er verfügbar ist.

## Erstellen eines privaten Gerätepools mit privaten Geräten (Konsole)

Wenn Sie einen Testlauf erstellen, können Sie einen Gerätepool für den Testlauf erstellen, und überprüfen, dass der Pool nur Ihre privaten Geräte umfasst.

### Note

Wenn Sie in der Konsole einen Gerätepool mit privaten Geräten erstellen, können Sie nur eine der drei verfügbaren Regeln für die Auswahl privater Geräte verwenden. Wenn Sie einen Gerätepool erstellen möchten, der mehrere Arten von Regeln für private Geräte enthält (z. B. Gerätepools, die Regeln für Geräte-ARNs und Geräteinstanz-ARNs enthalten), müssen Sie den Pool über die CLI oder API erstellen.

1. Öffnen Sie die Device Farm-Konsole unter <https://console.aws.amazon.com/devicefarm/>.

2. Wählen Sie im Device Farm-Navigationsbereich Testen von Mobilgeräten und wählen Sie dann Projekte.
3. Wählen Sie ein vorhandenes Projekt aus der Liste aus oder erstellen Sie ein neues. Um ein neues Projekt zu erstellen, wählen Sie Neues Projekt, geben Sie einen Namen für das Projekt ein, und wählen Sie dann Einreichen.
4. Wählen Sie Automated tests (Automatisierte Tests) aus und klicken Sie anschließend auf Create a new run (Einen neuen Testlauf erstellen). Der Assistent führt Sie durch die Schritte zum Auswählen Ihrer Anwendung und zur Konfiguration der Tests, die Sie ausführen möchten.
5. Für die Geräte auswählen Schritt, wähle Gerätepool erstellen und geben Sie einen Namen und eine optionale Beschreibung für Ihren Gerätepool ein.
  - a. Um ARN-Regeln für Geräte für Ihren Gerätepool zu verwenden, wählen Sie Erstellen Sie einen statischen Gerätepool und wählen Sie dann die spezifischen Gerätetypen aus der Liste aus, die Sie im Gerätepool verwenden möchten. Wählen Sie nicht Nur private Geräteinstanzen weil diese Option bewirkt, dass der Gerätepool mit ARN-Regeln für Geräteinstanzen (anstelle von Geräte-ARN-Regeln) erstellt wird.

**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. Um Labelregeln für Geräteinstanzen für Ihren Gerätepool zu verwenden, wählen Sie Erstellen Sie einen dynamischen Gerätepool. Wählen Sie dann für jedes Label, das Sie im Gerätepool verwenden möchten Fügen Sie eine Regel hinzu. Wählen Sie für jede Regel Instanzbezeichnungen als Field, wähle Enthält als Operator, und geben Sie das gewünschte Geräteinstanz-Label als Value.

**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 |

Add a rule

**Max devices**  
Enter max number of devices

If you do not enter the max devices, we will pick all devices in our fleet that match the above rules

**Mobile devices (0/92)**  
Find devices by attribute

| Name | Status | Platform | OS | Form factor | Instance Id | Labels |
|------|--------|----------|----|-------------|-------------|--------|
|      |        |          |    |             |             |        |

Cancel Create

- c. Um ARN-Regeln für Geräteinstanzen für Ihren Gerätepool zu verwenden, wählen Sie **Erstellen Sie einen statischen Gerätepool**, wählen Sie dann **Nur private Geräteinstanzen** um die Liste der Geräte auf nur die privaten Geräteinstanzen zu beschränken, die Device Farm mit Ihrem verknüpft hat **AWS** Konto.

**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

6. Wählen Sie **Erstellen** aus.

## Einen privaten Gerätepool mit privaten Geräten erstellen (AWS CLI)

- Führen Sie den Befehl [create-device-pool](#) aus.

Informationen zur Verwendung von Device Farm mit dem **AWS CLI**, siehe [AWS CLI-Referenz](#).

## Erstellen eines privaten Gerätepools mit privaten Geräten (API)

- Rufen Sie die [CreateDevicePool](#)-API auf.

Informationen zur Verwendung der Device Farm API finden Sie unter [Gerätefarm wird automatisiert](#).

## Überspringen der erneuten Signierung von Apps auf privaten Geräten in AWS Device Farm

Das Signieren von Apps ist ein Prozess, bei dem ein App-Paket (z. B. [APK](#), [IPA](#)) mit einem privaten Schlüssel digital signiert wird, bevor es auf einem Gerät installiert oder in einem App Store wie dem Google Play Store oder dem Apple App Store veröffentlicht werden kann. Um das Testen zu optimieren, indem die Anzahl der benötigten Signaturen und Profile reduziert und die Datensicherheit auf Remote-Geräten erhöht wird, signiert AWS Device Farm Ihre App erneut, nachdem sie in den Service hochgeladen wurde.

Sobald Sie Ihre App auf AWS Device Farm hochgeladen haben, generiert der Service mithilfe seiner eigenen Signaturzertifikate und Bereitstellungsprofile eine neue Signatur für die App. Dieser Prozess ersetzt die ursprüngliche App-Signatur durch die Signatur von AWS Device Farm. Die neu signierte App wird dann auf den von AWS Device Farm bereitgestellten Testgeräten installiert. Die neue Signatur ermöglicht die Installation und Ausführung der App auf diesen Geräten, ohne dass die ursprünglichen Entwicklerzertifikate erforderlich sind.

Auf iOS ersetzen wir das eingebettete Bereitstellungsprofil durch ein Platzhalterprofil und kündigen die App. Wenn Sie es angeben, fügen wir dem Anwendungspaket vor der Installation zusätzliche Daten hinzu, sodass die Daten in der Sandbox Ihrer App vorhanden sind. Wenn Sie die iOS-App kündigen, werden bestimmte Rechte entfernt. Dazu gehören App Group, zugehörige Domains, Game Center, HealthKit, HomeKit Konfiguration von drahtlosem Zubehör, In-App-Kauf, Inter-App-Audio, Apple Pay, Push-Benachrichtigungen sowie VPN-Konfiguration und -Steuerung.

Auf Android kündigen wir die App. Dadurch können Funktionen beeinträchtigt werden, die von der App-Signatur abhängen, z. B. die Google Maps Android API. Es kann auch die Erkennung von Piraterie und Manipulation auslösen, die in Produkten wie zum Beispiel verfügbar sind. DexGuard Bei integrierten Tests können wir das Manifest so ändern, dass es die erforderlichen Berechtigungen für die Erfassung und Speicherung von Screenshots enthält.

Wenn Sie private Geräte verwenden, können Sie den Schritt überspringen, bei dem AWS Device Farm Ihre App erneut signiert. Dies unterscheidet sich von öffentlichen Geräten, bei denen Device Farm Ihre App auf den Android- und iOS-Plattformen immer neu signiert.

Sie können die erneute App-Signatur überspringen, wenn Sie eine Remote-Zugriffssitzung oder einen Testlauf erstellen. Dies kann hilfreich sein, wenn Ihre App über Funktionen verfügt, die nicht mehr funktionieren, wenn Device Farm Ihre App erneut signiert. Beispielsweise funktionieren Push-Benachrichtigungen möglicherweise nicht mehr, nachdem eine erneute Signatur durchgeführt wurde. Weitere Informationen zu den Änderungen, die Device Farm beim Testen Ihrer App vornimmt, finden Sie in den [häufig gestellten Fragen zu AWS Device Farm](#) oder auf der [Apps-Seite](#).

Um die erneute Signatur der App für einen Testlauf zu überspringen, wählen Sie beim Erstellen des Testlaufs die Option Skip app re-signing (Überspringen der erneuten Signatur von Apps) auf der Seite Configure (Konfigurieren) aus.

# Configure

## Setup test framework

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

Built-in: Fuzz

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

### Event count

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

6000

### Event throttle

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

50

### Randomizer seed

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

*Enter a randomizer seed*

## ▼ Advanced Configuration (optional)

### Configuration specific to Private Devices

#### App re-signing

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

Skip app re-signing

#### Other Configuration

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

#### Video recording

If checked, enables video recording during test execution.

Enable video recording

## Note

Wenn Sie das XCTest-Framework verwenden, steht die Option Skip app re-signing (Überspringen der erneuten Signatur von Apps) nicht zur Verfügung. Weitere Informationen finden Sie unter [Arbeiten mit XCTest für iOS und AWS Device Farm](#).

Zusätzliche Schritte zum Konfigurieren Ihrer App-Signatureinstellungen variieren, je nachdem, ob Sie private Android- oder iOS-Geräte verwenden.

## Das erneute Signieren von Apps auf Android-Geräten wird übersprungen

Beim Testen Ihrer App auf einem privaten Android-Gerät, wählen Sie Skip app re-signing (Überspringen der erneuten Signatur von Apps), wenn Sie Ihren Testlauf oder Ihre Remote-Zugriffssitzung erstellen. Weitere Konfigurationsarbeiten sind nicht erforderlich.

## Das erneute Signieren von Apps auf iOS-Geräten wird übersprungen

Apple erfordert eine Signatur zum Testen der App, bevor Sie sie auf ein Gerät laden können. Für iOS-Geräte haben Sie zwei Möglichkeiten, Ihre App zu signieren.

- Wenn Sie ein internes Entwickler-Profil (Enterprise) verwenden, können Sie diesen Schritt überspringen und mit dem nächsten Abschnitt, [the section called “Erstellen Sie eine Fernzugriffssitzung, um Ihrer App zu vertrauen”](#), fortfahren.
- Wenn Sie ein Ad-hoc-Entwicklungsprofil für iOS-Apps verwenden, müssen Sie zunächst das Gerät bei Ihrem Apple-Entwicklerkonto registrieren und anschließend Ihr Bereitstellungsprofil mit dem privaten Gerät aktualisieren. Anschließend müssen Sie Ihre App mit dem aktualisierten Bereitstellungsprofil erneut signieren. Anschließend können Sie Ihre neu signierte App in Device Farm ausführen.

So registrieren Sie ein Gerät mit einem Ad-hoc-Entwicklungs-Bereitstellungsprofil für iOS-Apps

1. Melden Sie sich bei Ihrem Apple-Entwicklerkonto an.
2. Navigieren Sie zu dem Abschnitt Certificates, IDs and Profiles (Zertifikate, IDs und Profile) der Konsole.
3. Gehen Sie zu Devices (Geräte).
4. Registrieren Sie das Geräts bei Ihrem Apple-Entwicklerkonto. Um den Namen und die UDID des Geräts abzurufen, verwenden Sie den ListDeviceInstances Betrieb der Device Farm API.
5. Gehen Sie zu Ihrem Bereitstellungsprofil und wählen Sie Edit (Bearbeiten) aus.
6. Wählen Sie das Gerät aus der Liste aus.
7. Rufen Sie in Xcode Ihr aktualisiertes Bereitstellungsprofil ab und signieren Sie die App erneut.

Weitere Konfigurationsarbeiten sind nicht erforderlich. Sie können jetzt eine Remote-Zugriffssitzung oder einen Testlauf erstellen und dann Skip app re-signing (Überspringen der erneuten Signatur von Apps) auswählen.

## Eine Fernzugriffssitzung erstellen, um Ihrer iOS-App zu vertrauen

Wenn Sie ein internes Entwickler-Bereitstellungsprofil (Enterprise) verwenden, müssen Sie ein einmaliges Verfahren ausführen, um dem internen App-Entwicklerzertifikat auf jedem Ihrer Geräte zu vertrauen.

Dazu können Sie entweder die zu testende App auf dem privaten Gerät installieren oder Sie können eine "Dummy"-App installieren, die mit demselben Zertifikat signiert ist, wie die App, die Sie testen möchten. Die Installation einer Dummy-App, die mit demselben Zertifikat signiert wurde, bietet einen Vorteil. Sobald Sie dem Konfigurationsprofil oder dem Enterprise App-Entwickler vertrauen, werden allen Apps von diesem Entwickler auf dem privaten Gerät vertraut, bis Sie diese Apps löschen. Wenn Sie also eine neue Version der App hochladen, müssen Sie nicht erneut dem App-Entwickler vertrauen. Dies ist besonders nützlich, wenn Sie Testautomatisierungen ausführen und nicht bei jedem Testen Ihrer App eine Remote-Zugriffssitzung erstellen wollen.

Bevor Sie Ihre Fernzugriffssitzung starten, folgen Sie den Schritten unter [Erstellen eines Instance-Profils](#). So erstellen oder ändern Sie ein Instanzprofil in Device Farm. Fügen Sie im Instanzprofil die Bundle-ID der Test-App oder Dummy-App zur Einstellung Pakete von der Bereinigung ausschließen hinzu. Hängen Sie dann das Instanzprofil an die private Geräteinstanz an, um sicherzustellen, dass Device Farm diese App nicht vom Gerät entfernt, bevor ein neuer Testlauf gestartet wird. Auf diese Weise wird sichergestellt, dass Ihr Entwicklerzertifikat vertrauenswürdig bleibt.

Sie können die "Dummy"-App unter Verwendung einer Remote-Zugriffssitzung auf das Gerät hochladen, sodass Sie die App starten und dem Entwickler vertrauen können.

1. Befolgen Sie die Anweisungen unter [Erstellen Sie eine Sitzung](#), um eine Remote-Zugriffssitzung zu erstellen. Verwenden Sie dazu das soeben erstellte Instance-Profil des privaten Geräts. Wenn Sie Ihre Sitzung erstellen, achten Sie darauf, Skip app re-signing (Überspringen der erneuten Signatur von Apps) auszuwählen.

---

### 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**

Filtern Sie die Liste der Geräte nach privaten Geräten, indem Sie `Private device instances only` (Nur private Geräte-Instances) auswählen. So stellen Sie sicher, dass Sie ein privates Gerät mit dem richtigen Instance-Profil verwenden.

Stellen Sie sicher, dass Sie auch die Dummy-App oder die App, die Sie testen möchten, zur Einstellung Pakete von der Bereinigung ausschließen für das Instanzprofil hinzufügen, das an diese Instanz angehängt ist.

2. Wenn Ihre Remotesitzung gestartet wird, wählen Sie `Datei` auswählen, um eine Anwendung zu installieren, die Ihr internes Bereitstellungsprofil verwendet.
3. Starten Sie die App, die Sie gerade hochgeladen haben.
4. Folgen Sie den Anweisungen, um dem Entwickler-Zertifikat zu vertrauen.

Alle Apps von diesem "Configuration Profile" (Konfigurationsprofil)- oder Enterprise App-Entwickler sind jetzt auf diesem privaten Gerät vertrauenswürdig, bis Sie sie löschen.

## Mit Amazon VPC zusammenarbeitenAWSRegionen

Device Farm-Dienste befinden sich nur im Westen der USA (Oregon) (`us-west-2`) Region. Sie können Amazon Virtual Private Cloud (Amazon VPC) verwenden, um einen Service in Ihrer Amazon Virtual Private Cloud in einem anderen zu erreichenAWSRegion, die die Gerätefarm verwendet. Wenn sich Device Farm und Ihr Dienst in derselben Region befinden, finden Sie weitere Informationen unter [Verwendung von Amazon VPC-Endpunktservices mit Device Farm — Legacy \(nicht empfohlen\)](#).

Es gibt zwei Möglichkeiten, auf Ihre privaten Dienste zuzugreifen, die sich in einer anderen Region befinden. Wenn Sie Dienste in einer anderen Region haben, ist dies nicht der Fallus-west-2, können Sie VPC Peering verwenden, um die VPC dieser Region per Peering mit einer anderen VPC zu verbinden, die mit Device Farm in verbunden istus-west-2. Wenn Sie jedoch Dienste in mehreren Regionen anbieten, ermöglicht Ihnen ein Transit Gateway den Zugriff auf diese Dienste mit einer einfacheren Netzwerkkonfiguration.

Weitere Informationen finden Sie unter [VPC-Peering-Szenarien](#) in derLeitfaden für Amazon VPC Peering.

## VPC-Peering

Sie können zwei beliebige VPCs in verschiedenen Regionen miteinander verbinden, sofern sie unterschiedliche, sich nicht überlappende CIDR-Blöcke haben. Dadurch wird sichergestellt, dass alle privaten IP-Adressen eindeutig sind, und alle Ressourcen in den VPCs können sich gegenseitig adressieren, ohne dass irgendeine Form von Network Address Translation (NAT) erforderlich ist. Weitere Informationen zur CIDR-Notation finden Sie unter [RFC 4632](#).

Dieses Thema enthält ein regionsübergreifendes Beispielszenario, in dem Device Farm (bezeichnet als VPC-1) befindet sich im Westen der USA (Oregon) (us-west-2) Region. Die zweite VPC in diesem Beispiel (bezeichnet als VPC-2) befindet sich in einer anderen Region.

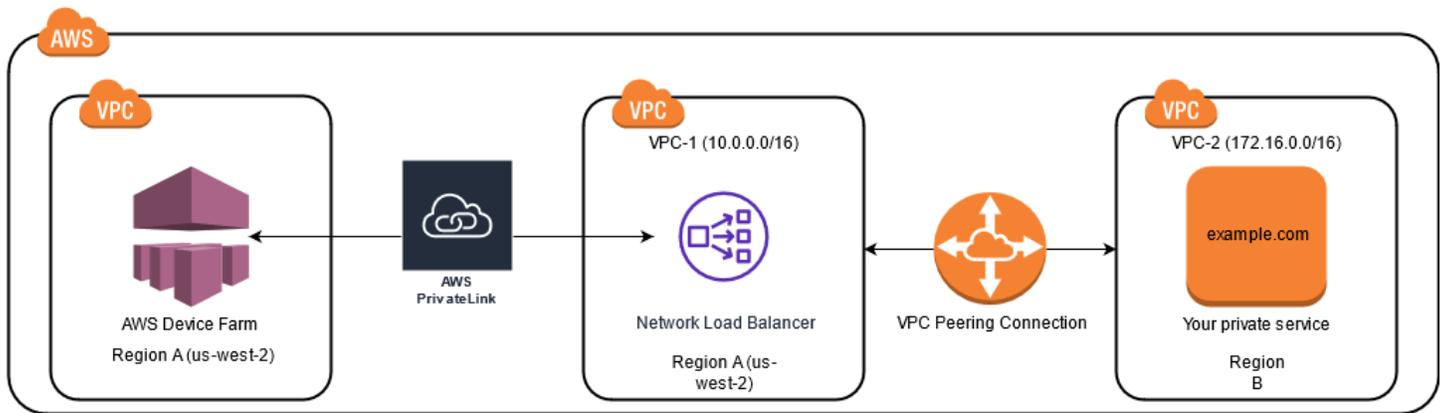
Beispiel für eine regionsübergreifende Device Farm VPC

| VPC-Komponente | VPC-1       | VPC-2         |
|----------------|-------------|---------------|
| CIDR           | 10.0.0.0/16 | 172.16.0.0/16 |

### Important

Das Herstellen einer Peering-Verbindung zwischen zwei VPCs kann den Sicherheitsstatus der VPCs ändern. Darüber hinaus kann das Hinzufügen neuer Einträge zu ihren Routing-Tabellen den Sicherheitsstatus der Ressourcen innerhalb der VPCs ändern. Es liegt in Ihrer Verantwortung, diese Konfigurationen so zu implementieren, dass sie den Sicherheitsanforderungen Ihres Unternehmens entsprechen. Weitere Informationen finden Sie auf der [Modell der geteilten Verantwortung](#).

Das folgende Diagramm zeigt die Komponenten dieses Beispiels und die Interaktionen zwischen diesen Komponenten.



## Themen

- [Voraussetzungen](#)
- [Schritt 1: Richten Sie eine Peering-Verbindung zwischen VPC-1 und VPC-2 ein](#)
- [Schritt 2: Aktualisieren Sie die Routentabellen in VPC-1 und VPC-2](#)
- [Schritt 3: Erstellen Sie eine Zielgruppe](#)
- [Schritt 4: Erstellen Sie einen Network Load Balancer](#)
- [Schritt 5: Erstellen Sie einen VPC-Endpunktdienst](#)
- [Schritt 6: Erstellen Sie eine VPC-Endpunktconfiguration in Device Farm](#)
- [Schritt 7: Erstellen Sie einen Testlauf](#)
- [Erstellen Sie mit Transit Gateway ein skalierbares Netzwerk](#)

## Voraussetzungen

Für dieses Beispiel müssen die folgenden Voraussetzungen erfüllt sein:

- Zwei VPCs, die mit Subnetzen konfiguriert sind, die sich nicht überlappende CIDR-Blöcke enthalten.
- VPC-1 muss in der us-west-2-Region und enthält Subnetze für Availability Zones us-west-2a, us-west-2b, und us-west-2c.

Weitere Informationen zum Erstellen von VPCs und zum Konfigurieren von Subnetzen finden Sie unter [Arbeiten mit VPCs und Subnetzen](#) in der Leitfaden für Amazon VPC Peering.

## Schritt 1: Richten Sie eine Peering-Verbindung zwischen VPC-1 und VPC-2 ein

Stellen Sie eine Peering-Verbindung zwischen den beiden VPCs her, die sich nicht überlappende CIDR-Blöcke enthalten. Informationen dazu finden Sie unter [VPC-Peering-Verbindungen erstellen und akzeptieren](#) in der Leitfaden für Amazon VPC Peering. Unter Verwendung des regionenübergreifenden Szenarios dieses Themas und des Leitfadens für Amazon VPC Peering, wird das folgende Beispiel für eine Peering-Verbindungskonfiguration erstellt:

### Name

Device-Farm-Peering-Connection-1

### VPC-ID (Anforderer)

vpc-0987654321gfedcba (VPC-2)

### Account

My account

### Region

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

### VPC-ID (Akzeptierer)

vpc-1234567890abcdefg (VPC-1)

### Note

Stellen Sie sicher, dass Sie beim Aufbau neuer Peering-Verbindungen Ihre VPC-Peering-Verbindungskontingente beachten. Weitere Informationen finden Sie unter [Amazon VPC-Kontingente](#) in der Leitfaden für Amazon VPC Peering.

## Schritt 2: Aktualisieren Sie die Routentabellen in VPC-1 und VPC-2

Nachdem Sie eine Peering-Verbindung eingerichtet haben, müssen Sie eine Zielroute zwischen den beiden VPCs einrichten, um Daten zwischen ihnen zu übertragen. Um diese Route einzurichten, können Sie die Routentabelle von manuell aktualisieren VPC-1 um auf das Subnetz von zu verweisen VPC-2 und umgekehrt. Um das zu tun, siehe [Aktualisieren Sie Ihre Routentabellen für](#)

[eine VPC-Peering-Verbindung](#) in der Leitfaden für Amazon VPC Peering. Unter Verwendung des regionenübergreifenden Szenarios dieses Themas und des Leitfadens für Amazon VPC Peering, die folgende Beispielkonfiguration für eine Routing-Tabelle wird erstellt:

Beispiel für eine VPC-Routentabelle für eine Gerätefarm

| VPC-Komponente        | VPC-1                 | VPC-2                 |
|-----------------------|-----------------------|-----------------------|
| Routing-Tabellen-ID   | rtb-1234567890abcdefg | rtb-0987654321gfedcba |
| Lokaler Adressbereich | 10.0.0.0/16           | 172.16.0.0/16         |
| Zieladressbereich     | 172.16.0.0/16         | 10.0.0.0/16           |

### Schritt 3: Erstellen Sie eine Zielgruppe

Nachdem Sie Ihre Zielrouten eingerichtet haben, können Sie einen Network Load Balancer in VPC-1 konfigurieren, um Anfragen weiterzuleiten an VPC-2.

Der Network Load Balancer muss zunächst eine Zielgruppe enthalten, die die IP-Adressen enthält, an die Anfragen gesendet werden.

Um eine Zielgruppe zu erstellen

1. Identifizieren Sie die IP-Adressen des Dienstes, auf den Sie abzielen möchten VPC-2.

- Diese IP-Adressen müssen Mitglieder des Subnetzes sein, das für die Peering-Verbindung verwendet wird.
- Die Ziel-IP-Adressen müssen statisch und unveränderlich sein. Wenn Ihr Dienst über dynamische IP-Adressen verfügt, sollten Sie erwägen, eine statische Ressource (z. B. einen Network Load Balancer) als Ziel zu verwenden und diese statische Ressource Anfragen an Ihr eigentliches Ziel weiterleiten zu lassen.

#### Note

- Wenn Sie auf eine oder mehrere eigenständige Amazon Elastic Compute Cloud (Amazon EC2) -Instances abzielen, öffnen Sie die Amazon EC2-Konsole unter <https://console.aws.amazon.com/ec2/>, dann wähle Instanzen.
- Wenn Sie auf eine Amazon EC2 Auto Scaling-Gruppe von Amazon EC2-Instances abzielen, müssen Sie die Amazon EC2 Auto Scaling-Gruppe einem Network

Load Balancer zuordnen. Weitere Informationen finden Sie unter [Einen Load Balancer zu Ihrer Auto Scaling-Gruppe hinzufügen](#) in der Amazon EC2 Auto Scaling-Benutzerhandbuch.

Anschließend können Sie die Amazon EC2-Konsole unter öffnen <https://console.aws.amazon.com/ec2/>, und wählen Sie dann Netzwerkschnittstellen. Von dort aus können Sie sich die IP-Adressen der einzelnen Netzwerkschnittstellen des Network Load Balancers anzeigen lassen Verfügbarkeitszone.

2. Erstellen Sie eine Zielgruppe in VPC-1. Um das zu tun, siehe [Erstellen Sie eine Zielgruppe für Ihren Network Load Balancer](#) in der Benutzerhandbuch für Network Load Balancer.

Zielgruppen für Dienste in einer anderen VPC erfordern die folgende Konfiguration:

- Für Wählen Sie einen Zieltyp, wählen IP-Adressen.
- Für VPC, wählen Sie die VPC aus, die den Load Balancer hosten soll. Für das Themenbeispiel wird dies sein VPC-1.
- Auf dem Ziele registrieren Seite, registrieren Sie ein Ziel für jede IP-Adresse in VPC-2.

Für Netzwerk, wählen Andere private IP-Adresse.

Für Verfügbarkeitszone, wählen Sie Ihre gewünschten Zonen in VPC-1.

Für IPv4-Adresse, wählen Sie die VPC-2 IP-Adresse.

Für Häfen, wählen Sie Ihre Anschlüsse.

- Wähle Weiter unten als ausstehend angeben. Wenn Sie mit der Angabe von Adressen fertig sind, wählen Sie Registrieren Sie ausstehende Ziele.

Unter Verwendung des regionenübergreifenden Szenarios dieses Themas und des Benutzerhandbuch für Network Load Balancer, die folgenden Werte werden in der Zielgruppenkonfiguration verwendet:

Zieltyp

IP addresses

Name der Zielgruppe

`my-target-group`

## Protokoll/Port

TCP : 80

## VPC

vpc-1234567890abcdefg (VPC-1)

## Netzwerk

Other private IP address

## Availability Zone

all

## IPv4-Adresse

172.16.100.60

## Anschlüsse

80

## Schritt 4: Erstellen Sie einen Network Load Balancer

Erstellen Sie einen Network Load Balancer mit der unter beschriebenen Zielgruppe [Schritt 3](#). Um das zu tun, siehe [Einen Network Load Balancer erstellen](#).

Im regionsübergreifenden Szenario dieses Themas werden die folgenden Werte in einer Beispielkonfiguration für den Network Load Balancer verwendet:

### Load balancer name

my-nlb

### Schema

Internal

### VPC

vpc-1234567890abcdefg (VPC-1)

### Zuweisung

us-west-2a - subnet-4i23iuufkdiufsloi

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

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

Protokoll/Port

```
TCP : 80
```

Zielgruppe

```
my-target-group
```

## Schritt 5: Erstellen Sie einen VPC-Endpunktdienst

Sie können den Network Load Balancer verwenden, um einen VPC-Endpunktdienst zu erstellen. Über diesen VPC-Endpunktdienst kann Device Farm eine Verbindung zu Ihrem Dienst herstellen in VPC-2 ohne zusätzliche Infrastruktur, wie z. B. ein Internet-Gateway, eine NAT-Instance oder eine VPN-Verbindung.

Informationen dazu finden Sie unter [Einen Amazon VPC-Endpunkt service erstellen](#).

## Schritt 6: Erstellen Sie eine VPC-Endpunkt Konfiguration in Device Farm

Jetzt können Sie eine private Verbindung zwischen Ihrer VPC und der Gerätefarm herstellen. Sie können Device Farm verwenden, um private Dienste zu testen, ohne sie über das öffentliche Internet verfügbar zu machen. Informationen dazu finden Sie unter [Erstellen einer VPC-Endpunkt Konfiguration in Device Farm](#).

Im regionsübergreifenden Szenario dieses Themas werden die folgenden Werte in einer Beispielformatierung für einen VPC-Endpunkt verwendet:

Name

```
My VPCE Configuration
```

Name des VPCE-Dienstes

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

DNS-Name des Dienstes

```
devicefarm.com
```

## Schritt 7: Erstellen Sie einen Testlauf

Sie können Testläufe erstellen, die die unter beschriebene VPC-Endpunktkonfiguration verwenden [Schritt 6](#). Weitere Informationen finden Sie unter [Erstellen Sie einen Testlauf in Device Farm](#) oder [Erstellen Sie eine Sitzung](#).

### Erstellen Sie mit Transit Gateway ein skalierbares Netzwerk

Um ein skalierbares Netzwerk mit mehr als zwei VPCs zu erstellen, können Sie Transit Gateway als Netzwerk-Transit-Hub verwenden, um Ihre VPCs und lokalen Netzwerke miteinander zu verbinden. Um eine VPC in derselben Region wie Device Farm für die Verwendung eines Transit Gateways zu konfigurieren, folgen Sie den [Amazon VPC-Endpunktdienste mit Device Farm](#) Leitfaden zur gezielten Nutzung von Ressourcen in einer anderen Region auf der Grundlage ihrer privaten IP-Adressen.

Weitere Informationen zu Transit Gateway finden Sie unter [Was ist ein Transit-Gateway?](#) in der Leitfaden für Amazon VPC Transit Gateways.

## Kündigung privater Geräte

### Important

Diese Anweisungen gelten nur für die Kündigung von Verträgen über private Geräte. Informationen zu allen anderen AWS Diensten und Fragen zur Abrechnung finden Sie in der jeweiligen Dokumentation für diese Produkte oder wenden Sie sich an den AWS Support.

<Um ein privates Gerät nach Ablauf der ursprünglich vereinbarten Laufzeit zu kün

# VPC-ENI in der AWS-Gerätefarm

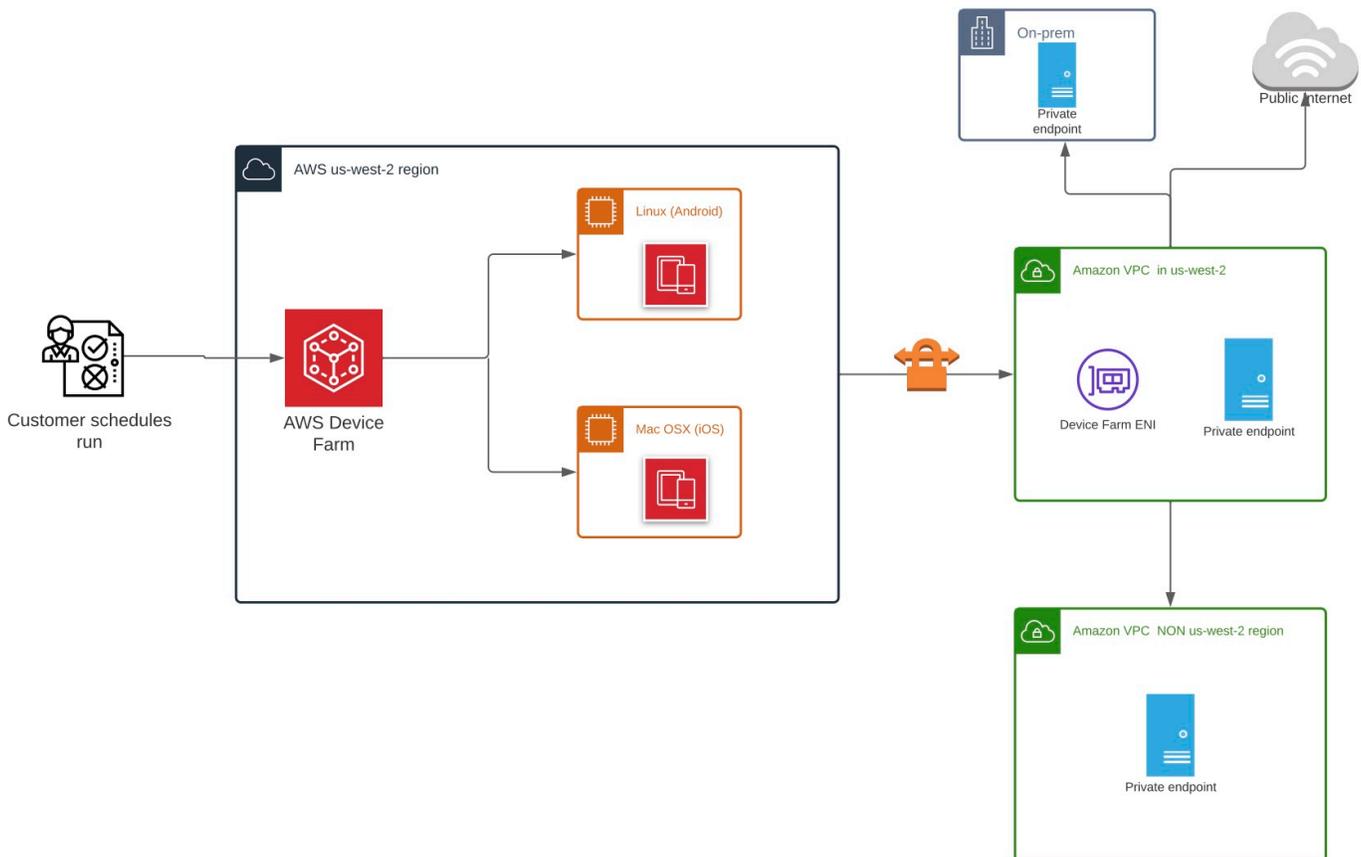
## Warning

[Diese Funktion ist nur auf privaten Geräten verfügbar.](#) Um die Nutzung privater Geräte auf Ihrem AWS Konto zu beantragen, [kontaktieren Sie uns](#) bitte. Wenn Sie Ihrem AWS Konto bereits private Geräte hinzugefügt haben, empfehlen wir dringend, diese Methode der VPC-Konnektivität zu verwenden.

Die VPC-ENI-Konnektivitätsfunktion von AWS Device Farm hilft Kunden dabei, eine sichere Verbindung zu ihren privaten Endpunkten herzustellen AWS, die auf einer lokalen Software oder einem anderen Cloud-Anbieter gehostet werden.

Sie können sowohl Device Farm Farm-Mobilgeräte als auch deren Host-Computer mit einer Amazon Virtual Private Cloud (Amazon VPC) -Umgebung in der us-west-2 Region verbinden, die den Zugriff auf isolierte non-internet-facing Dienste und Anwendungen über eine [elastic network interface](#) ermöglicht. Weitere Informationen zu VPCs finden Sie im [Amazon VPC-Benutzerhandbuch](#).

Wenn sich Ihr privater Endpunkt oder Ihre VPC nicht in der us-west-2 Region befindet, können Sie sie mithilfe von Lösungen wie einem [Transit Gateway](#) oder VPC-Peering mit einer [VPC](#) in der us-west-2 Region verknüpfen. In solchen Situationen erstellt Device Farm eine ENI in einem Subnetz, das Sie für Ihre us-west-2 Regions-VPC bereitstellen, und Sie sind dafür verantwortlich, dass eine Verbindung zwischen der us-west-2 Regions-VPC und der VPC in der anderen Region hergestellt werden kann.



Informationen zur automatischen Erstellung von VPCs und AWS CloudFormation zur Erstellung von Peering-Verbindungen finden Sie in den [VPC-Peering-Vorlagen im Vorlagen-Repository](#) unter [AWS CloudFormation GitHub](#)

### Note

Device Farm berechnet nichts für die Erstellung von ENIs in der VPC eines Kunden in us-west-2. Die Kosten für regionsübergreifende oder externe VPC-Inter-VPC-Konnektivität sind in dieser Funktion nicht enthalten.

Sobald Sie den VPC-Zugriff konfiguriert haben, können die Geräte und Hostmaschinen, die Sie für Ihre Tests verwenden, keine Verbindung zu Ressourcen außerhalb der VPC herstellen (z. B.

öffentliche CDNs), es sei denn, es gibt ein NAT-Gateway, das Sie innerhalb der VPC angeben. Weitere Informationen finden Sie unter [NAT-Gateways](#) im Amazon VPC-Benutzerhandbuch.

## Themen

- [AWS Zugriffskontrolle und IAM](#)
- [Service-verknüpfte Rollen](#)
- [Voraussetzungen](#)
- [Verbindung zu Amazon VPC herstellen](#)
- [Einschränkungen](#)
- [Verwendung von Amazon VPC-Endpunktservices mit Device Farm — Legacy \(nicht empfohlen\)](#)

## AWS Zugriffskontrolle und IAM

Mit AWS Device Farm können Sie [AWS Identity and Access Management](#)(IAM) Richtlinien erstellen, die den Zugriff auf die Funktionen von Device Farm gewähren oder einschränken. Um die VPC-Konnektivitätsfunktion mit AWS Device Farm zu verwenden, ist die folgende IAM-Richtlinie für das Benutzerkonto oder die Rolle erforderlich, die Sie für den Zugriff auf AWS Device Farm verwenden:

```
{
 "Version": "2012-10-17",
 "Statement": [{
 "Effect": "Allow",
 "Action": [
 "devicefarm:*",
 "ec2:DescribeVpcs",
 "ec2:DescribeSubnets",
 "ec2:DescribeSecurityGroups",
 "ec2:CreateNetworkInterface"
],
 "Resource": [
 "*"
]
 },
 {
 "Effect": "Allow",
 "Action": "iam:CreateServiceLinkedRole",
 "Resource": "arn:aws:iam::*:role/aws-service-role/devicefarm.amazonaws.com/AWSServiceRoleForDeviceFarm",
```

```
"Condition": {
 "StringLike": {
 "iam:AWSServiceName": "devicefarm.amazonaws.com"
 }
}
]
```

Um ein Device Farm Farm-Projekt mit einer VPC-Konfiguration zu erstellen oder zu aktualisieren, muss Ihre IAM-Richtlinie es Ihnen ermöglichen, die folgenden Aktionen für die in der VPC-Konfiguration aufgeführten Ressourcen aufzurufen:

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

Darüber hinaus muss Ihre IAM-Richtlinie auch die Erstellung der dienstbezogenen Rolle ermöglichen:

```
"iam:CreateServiceLinkedRole"
```

### Note

Keine dieser Berechtigungen ist für Benutzer erforderlich, die in ihren Projekten keine VPC-Konfigurationen verwenden.

## Service-verknüpfte Rollen

AWS Device Farm verwendet AWS Identity and Access Management (IAM) [serviceverknüpfte Rollen](#). Eine dienstverknüpfte Rolle ist ein einzigartiger Typ von IAM-Rolle, die direkt mit Device Farm verknüpft ist. Dienstbezogene Rollen sind von Device Farm vordefiniert und enthalten alle Berechtigungen, die der Dienst benötigt, um andere AWS Dienste in Ihrem Namen aufzurufen.

Eine dienstverknüpfte Rolle erleichtert die Einrichtung von Device Farm, da Sie die erforderlichen Berechtigungen nicht manuell hinzufügen müssen. Device Farm definiert die Berechtigungen seiner dienstbezogenen Rollen, und sofern nicht anders definiert, kann nur Device Farm seine Rollen

übernehmen. Die definierten Berechtigungen umfassen die Vertrauens- und Berechtigungsrichtlinie. Diese Berechtigungsrichtlinie kann keinen anderen IAM-Entitäten zugewiesen werden.

Sie können eine serviceverknüpfte Rolle erst löschen, nachdem ihre verwandten Ressourcen gelöscht wurden. Dadurch werden Ihre Device Farm Farm-Ressourcen geschützt, da Sie die Zugriffsberechtigung für die Ressourcen nicht versehentlich entfernen können.

Informationen zu anderen Services, die serviceverknüpfte Rollen unterstützen, finden Sie unter [AWS-Services, die mit IAM funktionieren](#). Suchen Sie nach den Services, für die Ja in der Spalte Serviceverknüpfte Rolle angegeben ist. Wählen Sie über einen Link Ja aus, um die Dokumentation zu einer serviceverknüpften Rolle für diesen Service anzuzeigen.

## Dienstbezogene Rollenberechtigungen für Device Farm

Device Farm verwendet die serviceverknüpfte Rolle mit dem Namen `AWSServiceRoleForDeviceFarm`— Erlaubt Device Farm, in Ihrem Namen auf AWS-Ressourcen zuzugreifen.

Die `AWSServiceRoleForDeviceFarm` serviceverknüpfte Rolle vertraut darauf, dass die folgenden Dienste die Rolle übernehmen:

- `devicefarm.amazonaws.com`

Die Rollenberechtigungsrichtlinie ermöglicht es Device Farm, die folgenden Aktionen auszuführen:

- Für Ihr Konto
  - Netzwerkschnittstellen erstellen
  - Beschreiben von Netzwerkschnittstellen
  - Beschreiben Sie VPCs
  - Beschreiben Sie Subnetze
  - Beschreiben von Sicherheitsgruppen
  - Schnittstellen löschen
  - Netzwerkschnittstellen ändern
- Für Netzwerkschnittstellen
  - Tags erstellen
- Für EC2-Netzwerkschnittstellen, die von Device Farm verwaltet werden
  - Berechtigungen für Netzwerkschnittstellen erstellen

## Die vollständige IAM-Richtlinie lautet:

```
{
 "Version": "2012-10-17",
 "Statement": [
 {
 "Effect": "Allow",
 "Action": [
 "ec2:DescribeNetworkInterfaces",
 "ec2:DescribeVpcs",
 "ec2:DescribeSubnets",
 "ec2:DescribeSecurityGroups"
],
 "Resource": "*"
 },
 {
 "Effect": "Allow",
 "Action": [
 "ec2:CreateNetworkInterface"
],
 "Resource": [
 "arn:aws:ec2:*:*:subnet/*",
 "arn:aws:ec2:*:*:security-group/*"
]
 },
 {
 "Effect": "Allow",
 "Action": [
 "ec2:CreateNetworkInterface"
],
 "Resource": [
 "arn:aws:ec2:*:*:network-interface/*"
],
 "Condition": {
 "StringEquals": {
 "aws:RequestTag/AWSDeviceFarmManaged": "true"
 }
 }
 },
 {
 "Effect": "Allow",
 "Action": [
 "ec2:CreateTags"
],
 },
```

```
"Resource": "arn:aws:ec2:*:*:network-interface/*",
"Condition": {
 "StringEquals": {
 "ec2:CreateAction": "CreateNetworkInterface"
 }
},
{
 "Effect": "Allow",
 "Action": [
 "ec2:CreateNetworkInterfacePermission",
 "ec2>DeleteNetworkInterface"
],
 "Resource": "arn:aws:ec2:*:*:network-interface/*",
 "Condition": {
 "StringEquals": {
 "aws:ResourceTag/AWSDeviceFarmManaged": "true"
 }
 }
},
{
 "Effect": "Allow",
 "Action": [
 "ec2:ModifyNetworkInterfaceAttribute"
],
 "Resource": [
 "arn:aws:ec2:*:*:security-group/*",
 "arn:aws:ec2:*:*:instance/*"
]
},
{
 "Effect": "Allow",
 "Action": [
 "ec2:ModifyNetworkInterfaceAttribute"
],
 "Resource": "arn:aws:ec2:*:*:network-interface/*",
 "Condition": {
 "StringEquals": {
 "aws:ResourceTag/AWSDeviceFarmManaged": "true"
 }
 }
}
]
```

Sie müssen Berechtigungen konfigurieren, damit eine juristische Stelle von IAM (z. B. Benutzer, Gruppe oder Rolle) eine serviceverknüpfte Rolle erstellen, bearbeiten oder löschen kann. Weitere Informationen finden Sie unter [serviceverknüpfte Rollenberechtigungen](#) im IAM-Benutzerhandbuch.

## Eine dienstverknüpfte Rolle für Device Farm erstellen

Wenn Sie eine VPC-Konfiguration für ein mobiles Testprojekt bereitstellen, müssen Sie eine serviceverknüpfte Rolle nicht manuell erstellen. Wenn Sie Ihre erste Device Farm-Ressource in der AWS Management Console, der AWS CLI oder der AWS API erstellen, erstellt Device Farm die dienstverknüpfte Rolle für Sie.

Wenn Sie diese serviceverknüpfte Rolle löschen und sie dann erneut erstellen müssen, können Sie dasselbe Verfahren anwenden, um die Rolle in Ihrem Konto neu anzulegen. Wenn Sie Ihre erste Device Farm-Ressource erstellen, erstellt Device Farm die dienstverknüpfte Rolle erneut für Sie.

Sie können die IAM-Konsole auch verwenden, um eine dienstverknüpfte Rolle mit dem Anwendungsfall Device Farm zu erstellen. Erstellen Sie in der AWS CLI oder der AWS API eine dienstverknüpfte Rolle mit dem `devicefarm.amazonaws.com` Dienstenamen. Weitere Informationen finden Sie unter [Erstellen einer serviceverknüpfte Rolle](#) im IAM-Leitfaden. Wenn Sie diese serviceverknüpfte Rolle löschen, können Sie mit demselben Verfahren die Rolle erneut erstellen.

## Bearbeiten einer dienstverknüpften Rolle für Device Farm

Device Farm erlaubt es Ihnen nicht, die `AWSServiceRoleForDeviceFarm` dienstverknüpfte Rolle zu bearbeiten. Da möglicherweise verschiedene Entitäten auf die Rolle verweisen, kann der Rollename nach dem Erstellen einer serviceverknüpften Rolle nicht mehr geändert werden. Sie können jedoch die Beschreibung der Rolle mit IAM bearbeiten. Weitere Informationen finden Sie unter [Bearbeiten einer serviceverknüpften Rolle](#) im IAM-Benutzerhandbuch.

## Löschen einer dienstverknüpften Rolle für Device Farm

Wenn Sie ein Feature oder einen Dienst, die bzw. der eine serviceverknüpften Rolle erfordert, nicht mehr benötigen, sollten Sie diese Rolle löschen. Auf diese Weise haben Sie keine ungenutzte juristische Stelle, die nicht aktiv überwacht oder verwaltet wird. Sie müssen jedoch die Ressourcen für Ihre serviceverknüpften Rolle zunächst bereinigen, bevor Sie sie manuell löschen können.

**Note**

Wenn der Device Farm Farm-Dienst die Rolle verwendet, wenn Sie versuchen, die Ressourcen zu löschen, schlägt das Löschen möglicherweise fehl. Wenn dies passiert, warten Sie einige Minuten und versuchen Sie es erneut.

So löschen Sie die serviceverknüpfte Rolle mit IAM

Verwenden Sie die IAM-Konsole, die oder die AWS API AWS CLI, um die `AWSServiceRoleForDeviceFarm` dienstverknüpfte Rolle zu löschen. Weitere Informationen finden Sie unter [Löschen einer serviceverknüpften Rolle](#) im IAM-Leitfaden.

## Unterstützte Regionen für mit Device Farm Farm-Dienst verknüpfte Rollen

Device Farm unterstützt die Verwendung von dienstbezogenen Rollen in allen Regionen, in denen der Dienst verfügbar ist. Weitere Informationen finden Sie unter [AWS -Regionen und Endpunkte](#).

Device Farm unterstützt nicht die Verwendung von dienstbezogenen Rollen in allen Regionen, in denen der Dienst verfügbar ist. Sie können die `AWSServiceRoleForDeviceFarm` Rolle in den folgenden Regionen verwenden.

| Name der Region            | Regions-ID     | Support in Device Farm |
|----------------------------|----------------|------------------------|
| USA Ost (Nord-Virginia)    | us-east-1      | Nein                   |
| USA Ost (Ohio)             | us-east-2      | Nein                   |
| USA West (Nordkalifornien) | us-west-1      | Nein                   |
| USA West (Oregon)          | us-west-2      | Ja                     |
| Asien-Pazifik (Mumbai)     | ap-south-1     | Nein                   |
| Asien-Pazifik (Osaka)      | ap-northeast-3 | Nein                   |
| Asien-Pazifik (Seoul)      | ap-northeast-2 | Nein                   |
| Asien-Pazifik (Singapur)   | ap-southeast-1 | Nein                   |

| Name der Region        | Regions-ID     | Support in Device Farm |
|------------------------|----------------|------------------------|
| Asien-Pazifik (Sydney) | ap-southeast-2 | Nein                   |
| Asien-Pazifik (Tokio)  | ap-northeast-1 | Nein                   |
| Kanada (Zentral)       | ca-central-1   | Nein                   |
| Europa (Frankfurt)     | eu-central-1   | Nein                   |
| Europa (Irland)        | eu-west-1      | Nein                   |
| Europa (London)        | eu-west-2      | Nein                   |
| Europa (Paris)         | eu-west-3      | Nein                   |
| Südamerika (São Paulo) | sa-east-1      | Nein                   |
| AWS GovCloud (US)      | us-gov-west-1  | Nein                   |

## Voraussetzungen

In der folgenden Liste werden einige Anforderungen und Vorschläge beschrieben, die Sie bei der Erstellung von VPC-ENI-Konfigurationen beachten sollten:

- Private Geräte müssen Ihrem AWS Konto zugewiesen werden.
- Sie müssen über einen AWS Kontobenutzer oder eine Rolle mit den erforderlichen Berechtigungen verfügen, um eine mit dem Dienst verknüpfte Rolle zu erstellen. Bei der Verwendung von Amazon VPC-Endpunkten mit den mobilen Testfunktionen von Device Farm erstellt Device Farm eine AWS Identity and Access Management (IAM) -Serviceverknüpfte Rolle.
- Device Farm kann nur eine Verbindung zu VPCs in der us-west-2 Region herstellen. Wenn Sie in der us-west-2 Region keine VPC haben, müssen Sie eine erstellen. Um dann auf Ressourcen in einer VPC in einer anderen Region zuzugreifen, müssen Sie eine Peering-Verbindung zwischen der VPC in der us-west-2 Region und der VPC in der anderen Region herstellen. Informationen zum Peering von VPCs finden Sie im [Amazon VPC Peering Guide](#).

Sie sollten bei der Konfiguration der Verbindung sicherstellen, dass Sie Zugriff auf Ihre angegebene VPC haben. Sie müssen bestimmte Amazon Elastic Compute Cloud (Amazon EC2) - Berechtigungen für Device Farm konfigurieren.

- In der VPC, die Sie verwenden, ist eine DNS-Auflösung erforderlich.
- Sobald Ihre VPC erstellt wurde, benötigen Sie die folgenden Informationen über die VPC in der `us-west-2` Region:
  - VPC-ID
  - Subnetz-IDs
  - Sicherheitsgruppen-IDs
- Sie müssen Amazon VPC-Verbindungen pro Projekt konfigurieren. Derzeit können Sie nur eine VPC-Konfiguration pro Projekt konfigurieren. Wenn Sie eine VPC konfigurieren, erstellt Amazon VPC eine Schnittstelle innerhalb Ihrer VPC und weist sie den angegebenen Subnetzen und Sicherheitsgruppen zu. Alle future Sitzungen, die mit dem Projekt verknüpft sind, werden die konfigurierte VPC-Verbindung verwenden.
- Sie können VPC-ENI-Konfigurationen nicht zusammen mit der älteren VPCE-Funktion verwenden.
- Wir empfehlen dringend, ein vorhandenes Projekt nicht mit einer VPC-ENI-Konfiguration zu aktualisieren, da bestehende Projekte möglicherweise VPCE-Einstellungen haben, die auf der Run-Ebene bestehen bleiben. Wenn Sie die vorhandenen VPCE-Funktionen bereits verwenden, verwenden Sie stattdessen VPC-ENI für alle neuen Projekte.

## Verbindung zu Amazon VPC herstellen

Sie können Ihr Projekt für die Verwendung von Amazon VPC-Endpunkten konfigurieren und aktualisieren. Die VPC-ENI-Konfiguration wird pro Projekt konfiguriert. Ein Projekt kann zu einem bestimmten Zeitpunkt nur einen VPC-ENI-Endpunkt haben. Um den VPC-Zugriff für ein Projekt zu konfigurieren, müssen Sie die folgenden Details kennen:

- Die VPC-ID in `us-west-2` wenn Ihre App dort gehostet wird, oder die `us-west-2` VPC-ID, die eine Verbindung zu einer anderen VPC in einer anderen Region herstellt.
- Die entsprechenden Sicherheitsgruppen, die auf die Verbindung angewendet werden sollen.
- Die Subnetze, die der Verbindung zugeordnet werden. Wenn eine Sitzung gestartet wird, wird das größte verfügbare Subnetz verwendet. Wir empfehlen, mehrere Subnetze verschiedenen Availability Zones zuzuordnen, um die Verfügbarkeit Ihrer VPC-Konnektivität zu verbessern.

Nachdem Sie Ihre VPC-ENI-Konfiguration erstellt haben, können Sie ihre Details mithilfe der Konsole oder CLI wie folgt aktualisieren.

## Console

1. Melden Sie sich bei der Device Farm Farm-Konsole unter <https://console.aws.amazon.com/devicefarm> an.
2. Wählen Sie im Navigationsbereich Device Farm die Option Mobile Device Testing und dann Projects aus.
3. Wählen Sie unter Mobile Testing-Projekte den Namen Ihres Projekts aus der Liste aus.
4. Wählen Sie Project settings (Projekteinstellungen) aus.
5. Im Bereich Virtual Private Cloud (VPC) -Einstellungen können Sie VPCSubnets, und Security Groups ändern.
6. Wählen Sie Speichern.

## CLI

Verwenden Sie den folgenden AWS-CLI-Befehl, um die Amazon VPC zu aktualisieren:

```
$ aws devicefarm update-project \
--arn arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef \
--vpc-config \
securityGroupIds=sg-02c1537701a7e3763,sg-005dadf9311efda25,\
subnetIds=subnet-09b1a45f9cac53717,subnet-09b1a45f9cac12345,\
vpcId=vpc-0238fb322af81a368
```

Sie können bei der Erstellung Ihres Projekts auch eine Amazon VPC konfigurieren:

```
$ aws devicefarm create-project \
--name VPCDemo \
--vpc-config \
securityGroupIds=sg-02c1537701a7e3763,sg-005dadf9311efda25,\
subnetIds=subnet-09b1a45f9cac53717,subnet-09b1a45f9cac12345,\
vpcId=vpc-0238fb322af81a368
```

# Einschränkungen

Die folgenden Einschränkungen gelten für die VPC-ENI-Funktion:

- Sie können bis zu fünf Sicherheitsgruppen in der VPC-Konfiguration eines Device Farm Farm-Projekts angeben.
- Sie können bis zu acht Subnetze in der VPC-Konfiguration eines Device Farm Farm-Projekts bereitstellen.
- Wenn Sie ein Device Farm Farm-Projekt für die Verwendung mit Ihrer VPC konfigurieren, muss das kleinste Subnetz, das Sie bereitstellen können, mindestens fünf verfügbare IPv4-Adressen haben.
- Öffentliche IP-Adressen werden derzeit nicht unterstützt. Stattdessen empfehlen wir, private Subnetze in Ihren Device Farm Farm-Projekten zu verwenden. Wenn Sie während Ihrer Tests einen öffentlichen Internetzugang benötigen, verwenden Sie ein [NAT-Gateway \(Network Address Translation\)](#). Durch die Konfiguration eines Device Farm Farm-Projekts mit einem öffentlichen Subnetz erhalten Ihre Tests weder Internetzugang noch eine öffentliche IP-Adresse.
- Nur ausgehender Datenverkehr von der dienstverwalteten ENI wird unterstützt. Dies bedeutet, dass die ENI keine unaufgeforderten eingehenden Anfragen von der VPC empfangen kann.

## Verwendung von Amazon VPC-Endpunktservices mit Device Farm — Legacy (nicht empfohlen)

### Warning

Wir empfehlen dringend, die auf [dieser](#) Seite beschriebene VPC-ENI-Konnektivität für private Endpunktkonnektivität zu verwenden, da VPCE jetzt als Legacy-Funktion gilt. VPC-ENI bietet mehr Flexibilität, einfachere Konfigurationen, ist kostengünstiger und erfordert im Vergleich zur VPCE-Konnektivitätsmethode deutlich weniger Wartungsaufwand.

### Note

Die Verwendung von Amazon VPC Endpoint Services mit Device Farm wird nur für Kunden mit konfigurierten privaten Geräten unterstützt. Um Ihrem AWS-Konto die Nutzung dieser Funktion mit privaten Geräten zu ermöglichen, [kontaktieren Sie uns](#) bitte.

Amazon Virtual Private Cloud (Amazon VPC) ist ein AWS Service, mit dem Sie AWS Ressourcen in einem von Ihnen definierten virtuellen Netzwerk starten können. Mit einer VPC haben Sie die Kontrolle über Ihre Netzwerkeinstellungen, z. B. den IP-Adressbereich, Subnetze, Routingtabellen und Netzwerk-Gateways.

Wenn Sie Amazon VPC verwenden, um private Anwendungen in der AWS Region USA West (Oregon) (us-west-2) zu hosten, können Sie eine private Verbindung zwischen Ihrer VPC und Device Farm herstellen. Mit dieser Verbindung können Sie Device Farm verwenden, um private Anwendungen zu testen, ohne sie über das öffentliche Internet verfügbar zu machen. [Wenden Sie sich an uns](#), damit Ihr AWS Konto diese Funktion auf privaten Geräten nutzen kann.

Um eine Ressource in Ihrer VPC mit Device Farm zu verbinden, können Sie die Amazon VPC-Konsole verwenden, um einen VPC-Endpunktservice zu erstellen. Mit diesem Endpunktdienst können Sie die Ressource in Ihrer VPC über einen Device Farm-VPC-Endpunkt für die Device Farm bereitstellen. Der Endpunktdienst bietet zuverlässige, skalierbare Konnektivität zu Device Farm, ohne dass ein Internet-Gateway, eine NAT-Instanz (Network Address Translation) oder eine VPN-Verbindung erforderlich ist. Weitere Informationen finden Sie unter [VPC Endpoint Services \(AWS PrivateLink\)](#) im AWS PrivateLink Handbuch.

#### Important

Die VPC-Endpunktfunktion von Device Farm hilft Ihnen dabei, private interne Dienste in Ihrer VPC mithilfe von Verbindungen sicher mit der öffentlichen Device Farm Farm-VPC zu verbinden. AWS PrivateLink Obwohl die Verbindung sicher und privat ist, hängt die Sicherheit vom Schutz Ihrer AWS -Anmeldeinformationen ab. Wenn Ihre AWS Anmeldeinformationen kompromittiert werden, kann ein Angreifer auf Ihre Servicedaten zugreifen oder diese der Außenwelt zugänglich machen.

Nachdem Sie einen VPC-Endpunkt-Service in Amazon VPC erstellt haben, können Sie die Device Farm-Konsole verwenden, um eine VPC-Endpunktkonfiguration in Device Farm zu erstellen. In diesem Thema erfahren Sie, wie Sie die Amazon VPC-Verbindung und die VPC-Endpunktkonfiguration in Device Farm erstellen.

## Bevor Sie beginnen

Die folgenden Informationen gelten für Amazon VPC-Benutzer in der Region USA West (Oregon) (us-west-2) mit einem Subnetz in jeder der folgenden Availability Zones: us-west-2a, us-west-2b und us-west-2c.

Device Farm stellt zusätzliche Anforderungen an die VPC-Endpunktdienste, mit denen Sie es verwenden können. Wenn Sie einen VPC-Endpunktdienst für die Verwendung mit Device Farm erstellen und konfigurieren, stellen Sie sicher, dass Sie Optionen auswählen, die die folgenden Anforderungen erfüllen:

- Die Availability Zones für den Service müssen us-west-2a, us-west-2b und us-west-2c umfassen. Der Network Load Balancer, der einem VPC-Endpunktdienst zugeordnet ist, bestimmt die Availability Zones für diesen VPC-Endpunktdienst. Wenn Ihr VPC-Endpunktdienst nicht alle drei Availability Zones anzeigt, müssen Sie Ihren Network Load Balancer neu erstellen, um diese drei Zonen zu aktivieren, und dann den Network Load Balancer wieder Ihrem Endpunktdienst zuordnen.
- Die zulässigen Prinzipale für den Endpunkt-Service müssen den Amazon-Ressourcennamen (ARN) des Device Farm Farm-VPC-Endpunkts (Service-ARN) enthalten. Nachdem Sie Ihren Endpunktdienst erstellt haben, fügen Sie den Device Farm VPC-Endpunktdienst-ARN zu Ihrer Zulassungsliste hinzu, um Device Farm die Erlaubnis zu erteilen, auf Ihren VPC-Endpunktdienst zuzugreifen. [Kontaktieren Sie uns](#), um den Device Farm VPC-Endpunktdienst ARN zu erhalten.

Wenn Sie bei der Erstellung Ihres VPC-Endpunktdienstes die Einstellung Akzeptanz erforderlich aktiviert lassen, müssen Sie außerdem jede Verbindungsanforderung, die Device Farm an den Endpunktdienst sendet, manuell akzeptieren. Um diese Einstellung für einen vorhandenen Endpunkt-Service zu ändern, wählen Sie den Endpunkt-Service auf der Amazon VPC-Konsole aus, wählen Sie Aktionen und dann Endpunkt-Akzeptanzeinstellung ändern. Weitere Informationen finden Sie im Leitfaden unter [Ändern der Load Balancer- und Akzeptanzeinstellungen](#).AWS PrivateLink

Im nächsten Abschnitt wird erklärt, wie Sie einen Amazon VPC-Endpunkt-service erstellen, der diese Anforderungen erfüllt.

## Schritt 1: Einen Network Load Balancer erstellen

Der erste Schritt beim Aufbau einer privaten Verbindung zwischen Ihrer VPC und Device Farm besteht darin, einen Network Load Balancer einzurichten, der Anfragen an eine Zielgruppe weiterleitet.

New console

So erstellen Sie einen Network Load Balancer mit der neuen Konsole

1. Öffnen Sie die Amazon Elastic Compute Cloud (Amazon EC2) -Konsole unter <https://console.aws.amazon.com/ec2/>.

2. Wählen Sie im Navigationsbereich unter Load Balancing die Option Load Balancers aus.
3. Wählen Sie Load Balancer erstellen aus.
4. Wählen Sie unter Network Load Balancer die Option Create aus.
5. Gehen Sie auf der Seite Network Load Balancer erstellen unter Grundkonfiguration wie folgt vor:
  - a. Geben Sie einen Namen für den Load Balancer ein.
  - b. Wählen Sie für Schema die Option Intern aus.
6. Führen Sie unter Network mapping (Netzwerk-Mapping) einen der folgenden Schritte aus:
  - a. Wählen Sie die VPC für Ihre Zielgruppe.
  - b. Wählen Sie die folgenden Mappings aus:
    - us-west-2a
    - us-west-2b
    - us-west-2c
7. Verwenden Sie unter Listener und Routing die Optionen Protokoll und Port, um Ihre Zielgruppe auszuwählen.

 Note

Standardmäßig ist der zonenübergreifende Lastenausgleich deaktiviert. Da der Load Balancer die Availability Zones verwendet us-west-2a us-west-2b, erfordert er entweder us-west-2c, dass Ziele in jeder dieser Availability Zones registriert sind, oder, wenn Sie Ziele in weniger als allen drei Zonen registrieren, müssen Sie zonenübergreifendes Load Balancing aktivieren. Andernfalls funktioniert der Load Balancer möglicherweise nicht wie erwartet.

8. Wählen Sie Load Balancer erstellen aus.

## Old console

So erstellen Sie einen Network Load Balancer mit der alten Konsole

1. Öffnen Sie die Amazon Elastic Compute Cloud (Amazon EC2) -Konsole unter <https://console.aws.amazon.com/ec2/>.

2. Wählen Sie im Navigationsbereich unter Load Balancing die Option Load Balancers aus.
3. Wählen Sie Load Balancer erstellen aus.
4. Wählen Sie unter Network Load Balancer die Option Create aus.
5. Gehen Sie auf der Seite Load Balancer konfigurieren unter Grundkonfiguration wie folgt vor:
  - a. Geben Sie einen Namen für den Load Balancer ein.
  - b. Wählen Sie für Schema die Option Intern aus.
6. Wählen Sie unter Listeners das Protokoll und den Port aus, die Ihre Zielgruppe verwendet.
7. Gehen Sie unter Availability Zones wie folgt vor:
  - a. Wählen Sie die VPC für Ihre Zielgruppe.
  - b. Wählen Sie die folgenden Availability Zones aus:
    - us-west-2a
    - us-west-2b
    - us-west-2c
  - c. Wählen Sie Weiter: Sicherheitseinstellungen konfigurieren.
8. (Optional) Konfigurieren Sie Ihre Sicherheitseinstellungen und wählen Sie dann Weiter: Routing konfigurieren.
9. Führen Sie auf der Seite Configure Routing (Weiterleitung konfigurieren) die folgenden Schritte aus:
  - a. Für Zielgruppe, wählen Sie Vorhandene Zielgruppe.
  - b. Wählen Sie unter Name Ihre Zielgruppe aus.
  - c. Wählen Sie Weiter: Ziele registrieren.
10. Überprüfen Sie auf der Seite Ziele registrieren Ihre Ziele und wählen Sie dann Weiter: überprüfen aus.

 Note

Standardmäßig ist der zonenübergreifende Load Balancing deaktiviert. Da der Load Balancer die Availability Zones verwendet us-west-2a us-west-2b, erfordert er entweder us-west-2c, dass Ziele in jeder dieser Availability Zones registriert sind, oder, wenn Sie Ziele in weniger als allen drei Zonen registrieren,

müssen Sie zonenübergreifendes Load Balancing aktivieren. Andernfalls funktioniert der Load Balancer möglicherweise nicht wie erwartet.

11. Überprüfen Sie Ihre Load Balancer-Konfiguration und wählen Sie dann Create aus.

## Schritt 2: Einen Amazon VPC-Endpunktservice erstellen

Nachdem Sie den Network Load Balancer erstellt haben, verwenden Sie die Amazon VPC-Konsole, um einen Endpunkt-Service in Ihrer VPC zu erstellen.

1. Öffnen Sie die Amazon VPC-Konsole unter <https://console.aws.amazon.com/vpc/>.
2. Wählen Sie unter Ressourcen nach Region die Option Endpoint Services aus.
3. Wählen Sie Create Endpoint Service (Endpunkt-Service erstellen) aus.
4. Führen Sie eine der folgenden Aktionen aus:
  - Wenn Sie bereits über einen Network Load Balancer verfügen, den der Endpoint Service verwenden soll, wählen Sie ihn unter Verfügbare Load Balancer aus, und fahren Sie dann mit Schritt 5 fort.
  - Wenn Sie noch keinen Network Load Balancer erstellt haben, wählen Sie Create new Load Balancer aus. Die Amazon EC2 EC2-Konsole wird geöffnet. Folgen Sie ab Schritt 3 den Schritten [unter Network Load Balancer erstellen](#) und fahren Sie dann mit diesen Schritten in der Amazon VPC-Konsole fort.
5. Vergewissern Sie sich, dass bei Inbegriffene Availability Zones us-west-2aus-west-2b, dass, und in der Liste us-west-2c erscheinen.
6. Wenn Sie nicht jede Verbindungsanfrage, die an den Endpoint Service gesendet wird, manuell annehmen oder ablehnen möchten, deaktivieren Sie unter Zusätzliche Einstellungen die Option Annahme erforderlich. Wenn Sie diese Option deaktivieren, akzeptiert der Endpunkt-Service automatisch alle Verbindungsanforderungen, die er empfängt.
7. Wählen Sie Erstellen.
8. Wählen Sie im neuen Endpunktdienst die Option Prinzipale zulassen aus.
9. [Kontaktieren Sie uns](#), um den ARN des Device Farm Farm-VPC-Endpunkts (Dienst-ARN) zu erhalten, der der Zulassungsliste für den Endpunktdienst hinzugefügt werden soll, und fügen Sie diesen Dienst-ARN dann der Zulassungsliste für den Dienst hinzu.

10. Notieren Sie sich auf der Registerkarte Details des Endpunkt-Services, den Namen des Services (Service-Name). Sie benötigen diesen Namen, wenn Sie die VPC-Endpunkt-Konfiguration im nächsten Schritt erstellen.

Ihr VPC-Endpunktdienst kann jetzt mit Device Farm verwendet werden.

### Schritt 3: Erstellen einer VPC-Endpunktkonfiguration in Device Farm

Nachdem Sie einen Endpunkt-Service in Amazon VPC erstellt haben, können Sie in Device Farm eine Amazon VPC-Endpunktkonfiguration erstellen.

1. Melden Sie sich bei der Device Farm Farm-Konsole unter <https://console.aws.amazon.com/devicefarm> an.
2. Wählen Sie im Navigationsbereich die Option Testen von Mobilgeräten und dann Private Geräte aus.
3. Wählen Sie VPCE-Konfigurationen aus.
4. Wählen Sie VPCE-Konfiguration erstellen.
5. Geben Sie unter Neue VPCE-Konfiguration erstellen einen Namen für die VPC-Endpunktkonfiguration ein.
6. Geben Sie als VPCE-Servicename den Namen des Amazon VPC-Endpunktdienstes (Servicename) ein, den Sie in der Amazon VPC-Konsole notiert haben. Das Name sieht wie folgt aus: `com.amazonaws.vpce.us-west-2.vpce-svc-id`.
7. Geben Sie unter Service-DNS-Name den Service-DNS-Namen für die App ein, die Sie testen möchten (z. B.). `devicefarm.com` Geben Sie vor dem Service-DNS-Namen weder `http` noch `https` an.

Der Domänenname ist nicht über das öffentliche Internet verfügbar. Darüber hinaus wird dieser neue Domainname, der Ihrem VPC-Endpunktservice zugeordnet ist, von Amazon Route 53 generiert und steht Ihnen in Ihrer Device Farm Farm-Sitzung exklusiv zur Verfügung.

8. Wählen Sie Speichern.

## 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

## Schritt 4: Einen Testlauf erstellen

Nachdem Sie die VPC-Endpunktkonfiguration gespeichert haben, können Sie die Konfiguration verwenden, um Testläufe oder Fernzugriffssitzungen zu erstellen. Weitere Informationen finden Sie unter [Erstellen Sie einen Testlauf in Device Farm](#) oder [Erstellen Sie eine Sitzung](#).

# Protokollieren von API-Aufrufen von AWS Device Farm mit AWS CloudTrail

AWS Device Farm ist integriert in AWS CloudTrail, ein Service, der eine Aufzeichnung der Aktionen bereitstellt, die von einem Benutzer, einer Rolle oder einem AWS Dienst in AWS Device Farm. CloudTrail erfasst alle API-Aufrufe für AWS Device Farm als Ereignisse. Zu den erfassten Aufrufen gehören Aufrufe von der AWS Device Farm-Konsole und Codeaufrufen für die API-Operationen von AWS Device Farm. Wenn Sie einen Trail erstellen, können Sie die kontinuierliche Bereitstellung von aktivierten CloudTrail Ereignissen in einem Amazon S3-Bucket, einschließlich Ereignissen für AWS Device Farm. Auch wenn Sie keinen Trail konfigurieren, können Sie die neuesten Ereignisse in der CloudTrail-Konsole in Event history (Ereignisverlauf) anzeigen. Unter Verwendung der gesammelten Informationen von CloudTrail, können Sie die Anfrage, die an AWS Device Farm gestellt wurde, die IP-Adresse, von der aus die Anfrage gestellt wurde, wer die Anfrage gestellt hat, wann sie gestellt wurde, und weitere Details ermitteln.

Weitere Informationen zu CloudTrail finden Sie im [AWS CloudTrail-Benutzerhandbuch](#).

## Informationen zur AWS-Gerätefarm in CloudTrail

CloudTrail wird beim Erstellen Ihres AWS-Kontos für Sie aktiviert. Wenn eine Aktivität in AWS Device Farm stattfindet, wird diese Aktivität in einem CloudTrail Ereignis zusammen mit anderen AWS Serviceveranstaltungen in Verlauf der Ereignisse. Sie können die neusten Ereignisse in Ihr AWS-Konto herunterladen und dort suchen und anzeigen. Weitere Informationen finden Sie unter [Anzeigen von Ereignissen mit dem CloudTrail -API-Ereignisverlauf](#).

Für eine fortlaufende Aufzeichnung der Ereignisse in Ihrem AWS-Konto, einschließlich Ereignissen für AWS Device Farm, erstellen Sie einen Trail. Ein Wanderweg aktiviert CloudTrail um Protokolldateien an einen Amazon S3-Bucket zu liefern. Wenn Sie einen Pfad in der Konsole anlegen, gilt dieser für alle AWS-Regionen. Der Trail protokolliert Ereignisse aus allen Regionen in der AWS-Partition und stellt die Protokolldateien in dem von Ihnen angegebenen Amazon S3 Bucket bereit. Darüber hinaus können Sie andere AWS-Services konfigurieren, um die in den CloudTrail-Protokollen erfassten Ereignisdaten weiter zu analysieren und entsprechend zu agieren. Weitere Informationen finden Sie hier:

- [Übersicht zum Erstellen eines Trails](#)
- [Von CloudTrail unterstützte Services und Integrationen](#)

- [Konfigurieren von Amazon SNS-Benachrichtigungen für CloudTrail](#)
- [Empfangen von CloudTrail-Protokolldateien aus mehreren Regionen](#) und [Empfangen von CloudTrail-Protokolldateien aus mehreren Konten](#)

Wann CloudTrail Die Protokollierung ist in Ihrem aktiviert AWS Konto, API-Aufrufe von Device Farm-Aktionen werden in Protokolldateien aufgezeichnet. Device Farm-Datensätze werden zusammen mit anderen geschriebenen AWS Serviceaufzeichnungen in einer Protokolldatei. Anhand eines Zeitraums und der Dateigröße bestimmt CloudTrail, wann diese Informationen in eine neue erstellte Datei geschrieben werden sollen.

Alle Aktionen der Device Farm werden protokolliert und dokumentiert in [AWS CLI-Referenz](#) und die [Gerätefarm wird automatisiert](#). Aufrufe zum Erstellen eines neuen Projekts oder zur Ausführung in Device Farm generieren beispielsweise Einträge in CloudTrail Protokolldateien.

Jeder Ereignis- oder Protokolleintrag enthält Informationen zu dem Benutzer, der die Anforderung generiert hat. Anhand der Identitätsinformationen zur Benutzeridentität können Sie Folgendes bestimmen:

- Ob die Anfrage mit Stammbenutzer- oder AWS Identity and Access Management (IAM)-Anmeldeinformationen ausgeführt wurde.
- Ob die Anforderung mit temporären Sicherheitsanmeldeinformationen für eine Rolle oder einen Verbundbenutzer ausgeführt wurde.
- Gibt an, ob die Anforderung aus einem anderen AWS-Service gesendet wurde

Weitere Informationen finden Sie unter [CloudTrail userIdentity-Element](#).

## Grundlegendes zu den Einträgen der AWS Device Farm-Protokolldateien

Ein Trail ist eine Konfiguration, durch die Ereignisse als Protokolldateien an den von Ihnen angegebenen Amazon-S3-Bucket übermittelt werden. CloudTrail-Protokolldateien können einen oder mehrere Einträge enthalten. Ein Ereignis stellt eine einzelne Anfrage aus einer beliebigen Quelle dar und enthält unter anderem Informationen über die angeforderte Aktion, das Datum und die Uhrzeit der Aktion sowie über die Anfrageparameter. CloudTrail-Protokolleinträge sind kein geordnetes Stacktrace der öffentlichen API-Aufrufe und erscheinen daher nicht in einer bestimmten Reihenfolge.

Das folgende Beispiel zeigt eine CloudTrail Protokolleintrag, der die Gerätefarm demonstriert ListRuns Aktion:

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

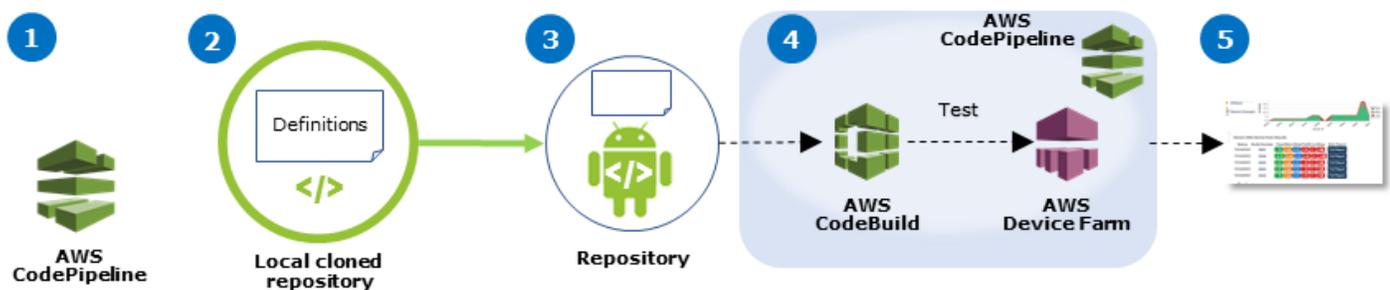
```
 "total": 4,
 "errored": 0
 },
 "type": "BUILTIN_FUZZ",
 "status": "RUNNING",
 "totalJobs": 3,
 "platform": "ANDROID_APP",
 "result": "PENDING"
 },
 ... additional entries ...
]
}
]
}
```

# Verwenden von AWS Device Farm in einem CodePipeline Testphase

Du kannst benutzen [AWS CodePipeline](#) um in Device Farm konfigurierte Tests für mobile Apps in eine von AWS verwaltete automatisierte Release-Pipeline zu integrieren. Sie können Ihre Pipeline so konfigurieren, dass Tests auf Aufforderung durchgeführt werden, nach einem festen Zeitplan oder als Teil eines stetigen Integrationsverlaufs.

Das folgende Diagramm zeigt den stetigen Integrationsverlauf, bei dem jedes Mal eine Android-App erstellt und getestet wird, wenn ein Push für ihr Repository bestätigt wird. Informationen zum Erstellen dieser Pipeline-Konfiguration finden Sie in [Tutorial: Erstellen und testen Sie eine Android-App, wenn Sie dazu gepusht werden GitHub](#).

Workflow to Set Up Android Application Test



1. Konfiguration

2. Definitionen  
hinzufügen

3. Push

4. Bauen  
und testen

5. Bericht

Pipeline-  
Ressourcen  
konfigurieren

Ihrem Paket  
Build- und  
Testdefinitionen  
hinzufügen

Ein Paket in  
Ihr Repositor  
y übertragen

Automatis  
ch ausgelöst  
es Erstellen  
und Testen  
von Apps aus  
Build-Aus  
gabeartefakten

Testergeb  
nisse anzeigen

Informationen zum Konfigurieren einer Pipeline, die ständig Tests einer kompilierten App (beispielsweise eine iOS- .ipa- oder Android- .apk-Datei) als deren Quelle durchführt, finden Sie unter [Tutorial: Testen einer iOS-App bei jedem Upload einer .ipa-Datei in einen Amazon S3-Bucket](#).

# konfigurierenCodePipelineum Ihre Device Farm-Tests zu verwenden

In diesen Schritten gehen wir davon aus, dass Sie [hat ein Device Farm-Projekt konfiguriert](#) und [hat eine Pipeline erstellt](#). Die Pipeline sollte mit einer Testphase konfiguriert sein, die ein [Eingangsartefakt erhält](#), das Ihre Testdefinition und kompilierte App-Paketdateien beinhaltet. Das Eingangsartefakt der Testphase kann das Ausgangsartefakt einer in Ihrer Pipeline konfigurierten Quell- oder Build-Phase sein.

Um einen Device Farm-Testlauf zu konfigurieren als CodePipelineTestaktion

1. Loggen Sie sich ein bei AWS Management Console und öffne das CodePipelineKonsole bei <https://console.aws.amazon.com/codepipeline/>.
2. Wählen Sie die Pipeline für Ihre App-Version aus.
3. Klicken Sie im Feld für die Testphase auf das Bleistiftsymbol und wählen Sie dann Action (Aktion) aus.
4. Klicken Sie im Bereich Add action (Aktion hinzufügen) für Action category (Aktionskategorie) auf Test (Testen).
5. Geben Sie unter Action name (Aktionsname) einen Namen ein.
6. Wählen Sie unter Test provider (Testanbieter) die Option AWS Device Farm aus.

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, there is a sub-prompt: 'Configure how your application is tested.' The 'Test actions' section is expanded, showing a list of test actions. The 'Action name\*' field is filled with 'test'. The 'Test provider\*' dropdown menu is set to 'AWS Device Farm'.

7. In Name des Projekts, wählen Sie Ihr vorhandenes Device Farm-Projekt oder wählen Erstellen Sie ein neues Projekt.

- Wählen Sie unter Device pool (Gerätepool) Ihren vorhandenen Gerätepool oder wählen Sie Create a new device pool (Einen neuen Gerätepool erstellen) aus. Wenn Sie einen Gerätepool erstellen, müssen Sie einen Satz Testgeräte auswählen.
- Wählen Sie unter App type (App-Typ) die Plattform für Ihre App aus.

### Device Farm Test

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

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

- Geben Sie unter App file path (App-Dateipfad) den Pfad des kompilierten App-Pakets ein. Der Pfad ist relativ zum Stamm des Eingabeartefakts für Ihren Test.
- Führen Sie unter Test type (Testtyp) einen der folgenden Schritte aus:
  - Wenn Sie einen der integrierten Device Farm-Tests verwenden, wählen Sie den in Ihrem Device Farm-Projekt konfigurierten Testtyp aus.
  - Wenn Sie keinen der integrierten Device Farm-Tests verwenden, finden Sie im Pfad der Testdatei, geben Sie den Pfad der Testspezifikationsdatei ein. Der Pfad ist relativ zum Stamm des Eingabeartefakts für Ihren Test.

The image shows three overlapping screenshots of the AWS Device Farm configuration interface for different test types:

- Top screenshot (Calabash):** Shows the 'Test type\*' dropdown set to 'Calabash' and the 'Test file path' text box containing 'tests.zip'. Below the text box is the label 'The location of artifact.'
- Middle screenshot (Appium Java TestNG):** Shows the 'Test type\*' dropdown set to 'Appium Java TestNG', the 'Test file path' text box containing 'tests.zip', the 'Appium version' text box containing '1.7.2', and a 'Use device slots' checkbox.
- Bottom screenshot (Built-in: Fuzz):** Shows the 'Test type\*' dropdown set to 'Built-in: Fuzz', the 'Event count' text box containing '6000', the 'Event throttle' text box containing '50', and the 'Randomizer seed' text box. Below the 'Event count' and 'Event throttle' text boxes are explanatory instructions: 'Specify a number between 1 and 10,000, representing the number of user interface events for the fuzz test to perform.' and '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.'

12. Geben Sie in den übrigen Feldern die Konfiguration ein, die für Ihren Test- und Anwendungstyp geeignet ist.
13. (Optional) Geben Sie unter Advanced (Erweitert) eine detaillierte Konfiguration für Ihren Testlauf ein.

▼ 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.

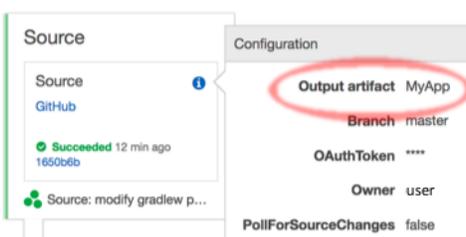
14. Wählen Sie in Input artifacts (Eingangsartefakte) das Eingangsartefakt aus, das mit dem Ausgangsartefakt der Phase übereinstimmt, die sich vor der Testphase in der Pipeline befindet.

**Input artifacts**

Choose one or more input artifacts for this action. The output of previous actions can be the input of this action. [Learn more](#)

**Input artifacts #1**

In der CodePipeline-Konsole finden Sie den Namen des Ausgangsartefakts für jede Phase, indem Sie mit der Maus auf das Informationssymbol im Pipeline-Diagramm zeigen. Wenn Ihre Pipeline Ihre App direkt von der QuelleBühne, wähleMyApp. Wenn Ihre Pipeline eine beinhaltetBauenBühne, wähleMyAppBuild.



15. Wählen Sie unten Add Action (Aktion hinzufügen) aus.
16. Wählen Sie im Bereich CodePipeline die Option Save pipeline change (Pipeline-Änderung speichern) und dann Save change (Änderung speichern).
17. Um Ihre Änderungen zu übertragen und einen Pipeline-Build zu starten, wählen Sie Release change (Änderung freigeben) und dann Release (Freigeben).

# AWS CLIREferenz für AWS Device Farm

Um die zu verwendenAWS Command Line Interface(AWS CLI) Informationen zum Ausführen von Device Farm-Befehlen finden Sie im[AWS CLIREferenz für AWS Device Farm](#).

Für allgemeine Informationen überAWS CLI, siehe[AWS Command Line InterfaceBenutzerleitfaden](#)und der[AWS CLIBefehlsreferenz](#).

# WindowsPowerShellReferenz für AWS Device Farm

Um Windows zu verwendenPowerShellInformationen zum Ausführen von Device Farm-Befehlen finden Sie im[Device Farm-Cmdlet-Referenz](#)in der[AWS Tools for Windows PowerShellCmdlet-Referenz](#). Weitere Informationen finden Sie unter[Einrichtung der AWS-Tools für WindowsPowerShell](#)in derAWS Tools for Windows PowerShellBenutzerleitfaden.

# Automatisieren der AWS-Gerätefarm

Der programmatische Zugriff auf Device Farm ist eine leistungsstarke Methode, um die allgemeinen Aufgaben zu automatisieren, die Sie erledigen müssen, wie z. B. das Planen eines Laufs oder das Herunterladen der Artefakte für einen Lauf, eine Suite oder einen Test. Dies ist mithilfe des AWS SDK und der AWS CLI möglich.

Die AWS SDK bietet Zugriff auf alle AWS Service, einschließlich Device Farm, Amazon S3 und mehr. Weitere Informationen finden Sie unter

- [AWS Tools und SDKs](#)
- die [API-Referenz für AWS Device Farm](#)

## Beispiel: Verwendung des AWS SDK zum Starten einer Gerätefarm, zum Ausführen und Sammeln von Artefakten

Das folgende Beispiel zeigt von Anfang bis Ende, wie Sie das verwenden können AWS SDK für die Arbeit mit Device Farm. Dieses Beispiel erledigt Folgendes:

- Lädt Test- und Anwendungspakete auf Device Farm hoch
- Startet einen Testlauf und wartet auf seinen Abschluss (oder einen Fehler)
- Lädt alle von den Testsuiten produzierten Artefakte herunter

Dieses Beispiel hängt für die Interaktion mit HTTP vom `requests`-Drittanbieterpaket ab.

```
import boto3
import os
import requests
import string
import random
import time
import datetime
import time
import json

The following script runs a test through Device Farm
#
```

```
Things you have to change:
config = {
 # This is our app under test.
 "appFilePath": "app-debug.apk",
 "projectArn": "arn:aws:devicefarm:us-
west-2:111122223333:project:1b99bcff-1111-2222-ab2f-8c3c733c55ed",
 # Since we care about the most popular devices, we'll use a curated pool.
 "testSpecArn": "arn:aws:devicefarm:us-west-2::upload:101e31e8-12ac-11e9-ab14-
d663bd873e83",
 "poolArn": "arn:aws:devicefarm:us-west-2::devicepool:082d10e5-d7d7-48a5-ba5c-
b33d66efa1f5",
 "namePrefix": "MyAppTest",
 # This is our test package. This tutorial won't go into how to make these.
 "testPackage": "tests.zip"
}

client = boto3.client('devicefarm')

unique =
 config['namePrefix']+"-"+(datetime.date.today().isoformat())+'.'.join(random.sample(string.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")
```

# Fehlerbehebung bei Device Farm-Fehlern

In diesem Abschnitt finden Sie Fehlermeldungen und Verfahren, die Ihnen helfen, häufige Probleme mit Device Farm zu beheben.

## Themen

- [Fehlerbehebung bei Android-Anwendungstests in AWS Device Farm](#)
- [Fehlerbehebung bei Appium Java JUnit-Tests in AWS Device Farm](#)
- [Fehlerbehebung bei Appium Java JUnit-Webanwendungstests in AWS Device Farm](#)
- [Fehlerbehebung bei Appium Java TestNG-Tests in AWS Device Farm](#)
- [Fehlerbehebung bei Appium Java TestNG-Webanwendungen in AWS Device Farm](#)
- [Fehlerbehebung bei Appium-Python-Tests in AWS Device Farm](#)
- [Fehlerbehebung bei Appium-Python-Webanwendungstests in AWS Device Farm](#)
- [Fehlerbehebung bei Instrumentierungstests in AWS Device Farm](#)
- [Fehlerbehebung bei iOS-Anwendungstests in AWS Device Farm](#)
- [Fehlerbehebung bei XCTest-Tests in AWS Device Farm](#)
- [Fehlerbehebung bei XCTest-UI-Tests in AWS Device Farm](#)

## Fehlerbehebung bei Android-Anwendungstests in AWS Device Farm

Im folgenden Thema werden Fehlermeldungen aufgeführt, die beim Hochladen von Android-Anwendungstests auftreten können, und Umgehungen für die einzelnen Fehler empfohlen.

### Note

Die folgenden Anweisungen gelten für Linux x86\_64 and Mac.

## ANDROID\_APP\_UNZIP\_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

**⚠ Warning**

Ihre Anwendung konnte nicht geöffnet werden. Prüfen Sie, ob die Datei gültig ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Anwendungspaket fehlerfrei dekomprimieren können. Im folgenden Beispiel ist der Name des Pakets `app-debug.apk`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip app-debug.apk
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Bei einem gültigen Android-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

```
.
|-- AndroidManifest.xml
|-- classes.dex
|-- resources.arsc
|-- assets (directory)
|-- res (directory)
`-- META-INF (directory)
```

Weitere Informationen finden Sie unter [Arbeiten mit Android-Tests in AWS Device Farm](#).

## ANDROID\_APP\_AAPT\_DEBUG\_BADGING\_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

**⚠ Warning**

Wir konnten keine Informationen zu Ihrer Anwendung extrahieren. Überprüfen Sie mit dem Befehl `aapt debug badging <path to your test package>`, ob die Anwendung gültig ist, und versuchen Sie es erneut, wenn der Befehl keine Fehler zurückgibt.

Während des Upload-Validierungsprozesses analysiert AWS Device Farm Informationen aus der Ausgabe eines `aapt debug badging <path to your package>`-Befehls.

Stellen Sie sicher, dass Sie diesen Befehl erfolgreich für Ihre Android-Anwendung ausführen können. Im folgenden Beispiel ist der Name des Pakets `app-debug.apk`.

- Kopieren Sie Ihr Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den Befehl aus:

```
$ aapt debug badging app-debug.apk
```

Bei einem gültigen Android-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

```
package: name='com.amazon.aws.adf.android.referenceapp' versionCode='1'
 versionName='1.0' platformBuildVersionName='5.1.1-1819727'
sdkVersion:'9'
application-label:'ReferenceApp'
application: label='ReferenceApp' icon='res/mipmap-mdpi-v4/ic_launcher.png'
application-debuggable
launchable-activity:
 name='com.amazon.aws.adf.android.referenceapp.Activities.MainActivity'
 label='ReferenceApp' icon=''
uses-feature: name='android.hardware.bluetooth'
uses-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'
```

Weitere Informationen finden Sie unter [Arbeiten mit Android-Tests in AWS Device Farm](#).

## ANDROID\_APP\_PACKAGE\_NAME\_VALUE\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Der Paketname konnte in Ihrer Anwendung nicht gefunden werden. Bitte stellen Sie sicher, dass die Anwendung gültig ist, indem Sie den Befehl `aapt debug badging <path to your test package>` ausführen, und versuchen Sie es erneut, nachdem der Paketname hinter dem Schlüsselwort „Paket: Name“ angezeigt wurde.

Während des Upload-Validierungsprozesses analysiert AWS Device Farm den Wert des Paketnamens aus der Ausgabe eines `aapt debug badging <path to your package>` Befehls.

Stellen Sie sicher, dass Sie diesen Befehl für Ihre Android-Anwendung ausführen und den Wert für den Paketnamen erfolgreich finden können. Im folgenden Beispiel ist der Name des Pakets `app-debug.apk`.

- Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ aapt debug badging app-debug.apk | grep "package: name="
```

Bei einem gültigen Android-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

```
package: name='com.amazon.aws.adf.android.referenceapp' versionCode='1'
versionName='1.0' platformBuildVersionName='5.1.1-1819727'
```

Weitere Informationen finden Sie unter [Arbeiten mit Android-Tests in AWS Device Farm](#).

## ANDROID\_APP\_SDK\_VERSION\_VALUE\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

**⚠ Warning**

Die SDK-Version konnte in Ihrer Anwendung nicht gefunden werden. Bitte stellen Sie sicher, dass die Anwendung gültig ist, indem Sie den Befehl `aapt debug badging <path to your test package>` ausführen, und versuchen Sie es erneut, nachdem die SDK-Version hinter dem Schlüsselwort `sdkVersion` angezeigt wurde.

Während des Upload-Validierungsprozesses analysiert AWS Device Farm den SDK-Versionswert aus der Ausgabe eines `aapt debug badging <path to your package>` Befehls.

Stellen Sie sicher, dass Sie diesen Befehl für Ihre Android-Anwendung ausführen und den Wert für den Paketnamen erfolgreich finden können. Im folgenden Beispiel ist der Name des Pakets `app-debug.apk`.

- Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ aapt debug badging app-debug.apk | grep "sdkVersion"
```

Bei einem gültigen Android-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

```
sdkVersion:'9'
```

Weitere Informationen finden Sie unter [Arbeiten mit Android-Tests in AWS Device Farm](#).

## ANDROID\_APP\_AAPT\_DUMP\_XMLTREE\_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

**⚠ Warning**

Wir konnten das gültige nicht finden `AndroidManifest.xml` in Ihrer Anwendung. Überprüfen Sie mit dem Befehl `aapt dump xmltree <path to your test package> AndroidManifest.xml`, ob das Testpaket gültig ist, und versuchen Sie es erneut, wenn der Befehl keine Fehler mehr zurückgibt.

Während des Upload-Validierungsprozesses analysiert AWS Device Farm mithilfe des folgenden Befehls Informationen aus dem XML-Analysebaum nach einer im Paket enthaltenen XML-Datei `aapt dump xmltree <path to your package> AndroidManifest.xml`.

Stellen Sie sicher, dass Sie diesen Befehl erfolgreich für Ihre Android-Anwendung ausführen können. Im folgenden Beispiel ist der Name des Pakets `app-debug.apk`.

- Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ aapt dump xmltree app-debug.apk. AndroidManifest.xml
```

Bei einem gültigen Android-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

```
N: android=http://schemas.android.com/apk/res/android
 E: manifest (line=2)
 A: android:versionCode(0x0101021b)=(type 0x10)0x1
 A: android:versionName(0x0101021c)="1.0" (Raw: "1.0")
 A: package="com.amazon.aws.adf.android.referenceapp" (Raw:
"com.amazon.aws.adf.android.referenceapp")
 A: platformBuildVersionCode=(type 0x10)0x16 (Raw: "22")
 A: platformBuildVersionName="5.1.1-1819727" (Raw: "5.1.1-1819727")
 E: uses-sdk (line=7)
 A: android:minSdkVersion(0x0101020c)=(type 0x10)0x9
 A: android:targetSdkVersion(0x01010270)=(type 0x10)0x16
 E: uses-permission (line=11)
 A: android:name(0x01010003)="android.permission.INTERNET" (Raw:
"android.permission.INTERNET")
 E: uses-permission (line=12)
 A: android:name(0x01010003)="android.permission.CAMERA" (Raw:
"android.permission.CAMERA")
```

Weitere Informationen finden Sie unter [Arbeiten mit Android-Tests in AWS Device Farm](#).

## ANDROID\_APP\_DEVICE\_ADMIN\_PERMISSIONS

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

**⚠ Warning**

Wir haben festgestellt, dass Ihre Anwendung Administratorberechtigungen für das Gerät erfordert. Bitte stellen Sie sicher, dass die Berechtigungen nicht erforderlich sind, indem Sie den Befehl `aapt dump xmltree <path to your test package> AndroidManifest.xml` ausführen, und versuchen Sie es erneut, nachdem Sie sichergestellt haben, dass die Ausgabe das Stichwort `android.permission.BIND_DEVICE_ADMIN` nicht enthält.

Während des Upload-Validierungsprozesses analysiert AWS Device Farm mithilfe des folgenden Befehls die Berechtigungsinformationen aus dem XML-Analysebaum für eine im Paket enthaltene XML-Datei `aapt dump xmltree <path to your package> AndroidManifest.xml`.

Stellen Sie sicher, dass Ihre Anwendung keine Administratorberechtigung für das Gerät benötigt. Im folgenden Beispiel ist der Name des Pakets `app-debug.apk`.

- Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ aapt dump xmltree app-debug.apk AndroidManifest.xml
```

Die Ausgabe sollte folgendermaßen aussehen:

```
N: android=http://schemas.android.com/apk/res/android
E: manifest (line=2)
 A: android:versionCode(0x0101021b)=(type 0x10)0x1
 A: android:versionName(0x0101021c)="1.0" (Raw: "1.0")
 A: package="com.amazonaws.devicefarm.android.referenceapp" (Raw:
"com.amazonaws.devicefarm.android.referenceapp")
 A: platformBuildVersionCode=(type 0x10)0x16 (Raw: "22")
 A: platformBuildVersionName="5.1.1-1819727" (Raw: "5.1.1-1819727")
E: uses-sdk (line=7)
 A: android:minSdkVersion(0x0101020c)=(type 0x10)0xa
 A: android:targetSdkVersion(0x01010270)=(type 0x10)0x16
E: uses-permission (line=11)
 A: android:name(0x01010003)="android.permission.INTERNET" (Raw:
"android.permission.INTERNET")
E: uses-permission (line=12)
```

```
A: android:name(0x01010003)="android.permission.CAMERA" (Raw:
"android.permission.CAMERA")
.....
```

Wenn die Android-Anwendung gültig ist, sollte die Ausgabe Folgendes nicht enthalten: A: android:name(0x01010003)="android.permission.BIND\_DEVICE\_ADMIN" (Raw: "android.permission.BIND\_DEVICE\_ADMIN").

Weitere Informationen finden Sie unter [Arbeiten mit Android-Tests in AWS Device Farm](#).

## Bestimmte Fenster in meiner Android-Anwendung zeigen einen leeren oder schwarzen Bildschirm

Wenn Sie eine Android-Anwendung testen und feststellen, dass bestimmte Fenster in der Anwendung in der Videoaufzeichnung Ihres Tests von Device Farm mit einem schwarzen Bildschirm angezeigt werden, verwendet Ihre Anwendung möglicherweise die von `AndroidFLAG_SECURE` Funktion. Diese Flagge (wie beschrieben in [Die offizielle Dokumentation von Android](#)) wird verwendet, um zu verhindern, dass bestimmte Fenster einer Anwendung mit Bildschirmaufzeichnungstools aufgezeichnet werden. Daher zeigt die Bildschirmaufzeichnungsfunktion von Device Farm (sowohl für Automatisierungs- als auch für Fernzugriffstests) möglicherweise einen schwarzen Bildschirm anstelle des Fensters Ihrer Anwendung an, wenn das Fenster diese Markierung verwendet.

Dieses Kennzeichen wird häufig von Entwicklern für Seiten in ihrer Anwendung verwendet, die vertrauliche Informationen enthalten, z. B. Anmeldeseiten. Wenn Sie für bestimmte Seiten wie die Anmeldeseite Ihrer Anwendung einen schwarzen Bildschirm anstelle des Bildschirms Ihrer Anwendung sehen, arbeiten Sie mit Ihren Entwicklern zusammen, um einen Build der Anwendung zu erhalten, der dieses Kennzeichen nicht zum Testen verwendet.

Beachten Sie außerdem, dass Device Farm immer noch mit Anwendungsfenstern interagieren kann, die über dieses Flag verfügen. Wenn also die Anmeldeseite Ihrer Anwendung als schwarzer Bildschirm angezeigt wird, können Sie möglicherweise immer noch Ihre Anmeldeinformationen eingeben, um sich bei der Anwendung anzumelden (und so Seiten zu sehen, die nicht blockiert sind von `FLAG_SECURE` Flagge).

# Fehlerbehebung bei Appium Java JUnit-Tests in AWS Device Farm

Im folgenden Thema werden Fehlermeldungen aufgeführt, die beim Hochladen von Appium Java JUnit-Tests auftreten können, und Behelfslösungen für die einzelnen Fehler empfohlen.

## Note

Die folgenden Anweisungen gelten für Linux x86\_64 and Mac.

## APPIUM\_JAVA\_JUNIT\_TEST\_PACKAGE\_PACKAGE\_UNZIP\_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

## Warning

Ihre ZIP-Datei für den Test konnte nicht geöffnet werden. Prüfen Sie, ob die Datei gültig ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Paketszip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Bei einem gültigen Appium Java JUnit-Paket sollte die Ausgabe wie folgt aussehen:

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
```

```
|– acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
everything built from the ./src/test directory)
|– zip-with-dependencies.zip (this .zip file contains all of the items)
`– dependency-jars (this is the directory that contains all of your dependencies,
built as JAR files)
 |– com.some-dependency.bar-4.1.jar
 |– com.another-dependency.thing-1.0.jar
 |– joda-time-2.7.jar
 `– log4j-1.2.14.jar
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_JAVA\_JUNIT\_TEST\_PACKAGE\_DEPENDENCY\_DIR\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Das Verzeichnis `dependency-jars` konnte in Ihrem Testpaket nicht gefunden werden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob das Verzeichnis `dependency-jars` in dem Paket enthalten ist, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets `zip-with-dependencies.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das Appium Java JUnit-Paket gültig ist, befindet sich das Verzeichnis *dependency-jars* im Arbeitsverzeichnis:

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
 built as JAR files)
 |- com.some-dependency.bar-4.1.jar
 |- com.another-dependency.thing-1.0.jar
 |- joda-time-2.7.jar
 `- log4j-1.2.14.jar
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_JAVA\_JUNIT\_TEST\_PACKAGE\_JAR\_MISSING\_IN\_DEPENDENCY\_DIR

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

In der Verzeichnisstruktur von „dependency-jars“ konnte keine JAR-Datei gefunden werden. Extrahieren Sie Ihr Testpaket und öffnen Sie dann das Verzeichnis „dependency-jars“; überprüfen Sie, ob sich im Verzeichnis mindestens eine JAR-Datei befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets zip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das Appium Java JUnit-Paket gültig ist, befindet sich mindestens eine *.jar*-Datei im Verzeichnis *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
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_JAVA\_JUNIT\_TEST\_PACKAGE\_TESTS\_JAR\_FILE\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

In Ihrem Testpaket konnte keine „\*-tests.jar“-Datei gefunden werden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob mindestens eine „\*-tests.jar“ in dem Paket enthalten ist, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Paketszip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das Appium Java JUnit-Paket gültig ist, finden Sie mindestens eine *Krug* Datei wie *acme-android-appium-1.0-Snapshot-Tests.jar* in unserem Beispiel. Bei anderen Projekten lautet der Dateiname anders, er sollte jedoch in jedem Fall mit *-tests.jar* enden.

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
 built as JAR files)
 |- com.some-dependency.bar-4.1.jar
 |- com.another-dependency.thing-1.0.jar
 |- joda-time-2.7.jar
 `- log4j-1.2.14.jar
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_JAVA\_JUNIT\_TEST\_PACKAGE\_CLASS\_FILE\_MISSING\_IN\_TESTS\_J

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

In der JAR-Datei der Tests konnte keine Klassendatei gefunden werden. Extrahieren Sie Ihr ZIP-Testpaket und dekomprimieren Sie dann die JAR-Datei für die Tests; überprüfen Sie, ob sich in der JAR-Datei mindestens eine Klassendatei befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets `zip-with-dependencies.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Du solltest mindestens eine JAR-Datei finden wie *acme-android-appium-1.0-Snapshot-Tests.jar* in unserem Beispiel. Bei anderen Projekten lautet der Dateiname anders, er sollte jedoch in jedem Fall mit *-tests.jar* enden.

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
 built as JAR files)
 |- com.some-dependency.bar-4.1.jar
 |- com.another-dependency.thing-1.0.jar
 |- joda-time-2.7.jar
 `- log4j-1.2.14.jar
```

3. Nachdem Sie die Dateien erfolgreich extrahiert haben, sollte in der Struktur des Arbeitsverzeichnisses bei der Ausführung des folgenden Befehls mindestens eine Klasse angezeigt werden:

```
$ tree .
```

Sie sollten eine Ausgabe wie die Folgende sehen:

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
```

```
| - one-class-file.class
|- folder
| ` - another-class-file.class
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
 built as JAR files)
 |- com.some-dependency.bar-4.1.jar
 |- com.another-dependency.thing-1.0.jar
 |- joda-time-2.7.jar
 ` - log4j-1.2.14.jar
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_JAVA\_JUNIT\_TEST\_PACKAGE\_JUNIT\_VERSION\_VALUE\_UNKNOWN

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Es konnte kein Wert für die JUnit-Version gefunden werden. Extrahieren Sie Ihr Testpaket und öffnen Sie das Verzeichnis „dependency-jars“; überprüfen Sie, ob sich im Verzeichnis mindestens eine JUnit-JAR-Datei befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Paketszip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
tree .
```

Die Ausgabe sollte in etwa wie folgt aussehen:

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
 built as JAR files)
 |- junit-4.10.jar
 |- com.some-dependency.bar-4.1.jar
 |- com.another-dependency.thing-1.0.jar
 |- joda-time-2.7.jar
 `- log4j-1.2.14.jar
```

Wenn das Appium Java JUnit-Paket gültig ist, enthält es in unserem Beispiel mindestens eine JUnit-Abhängigkeitsdatei ähnlich der Datei `junit-4.10.jar`. Der Dateiname setzt sich dabei aus dem Schlüsselwort `junit`, gefolgt von der Versionsnummer (in diesem Beispiel 4.10) zusammen.

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_JAVA\_JUNIT\_TEST\_PACKAGE\_INVALID\_JUNIT\_VERSION

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Es wurde erkannt, dass JUnit in einer Vorversion von 4.10, der minimalen unterstützten Version, verwendet wird. Bitte ändern Sie die JUnit-Version und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets `zip-with-dependencies.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

In dem Verzeichnis sollte eine JUnit-Abhängigkeitsdatei (in unserem Beispiel *junit-4.10.jar*) mit der richtigen Versionsnummer (im Beispiel 4.10) enthalten sein:

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
 built as JAR files)
 |- junit-4.10.jar
 |- com.some-dependency.bar-4.1.jar
 |- com.another-dependency.thing-1.0.jar
 |- joda-time-2.7.jar
 `- log4j-1.2.14.jar
```

#### Note

Ihre Tests werden möglicherweise nicht ordnungsgemäß ausgeführt, wenn die für JUnit angegebene Version in Ihrem Testpaket eine Vorversion der minimal unterstützten Version 4.10 ist.

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

# Fehlerbehebung bei Appium Java JUnit-Webanwendungstests in AWS Device Farm

Im folgenden Thema werden Fehlermeldungen aufgelistet, die beim Hochladen von Appium JUnit-Tests für Webanwendungen auftreten können, und Behelfslösungen für die einzelnen Fehler empfohlen. Weitere Informationen zur Verwendung von Appium mit Device Farm finden Sie unter [the section called “Appium”](#).

## APPIUM\_WEB\_JAVA\_JUNIT\_TEST\_PACKAGE\_UNZIP\_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Ihre ZIP-Datei für den Test konnte nicht geöffnet werden. Prüfen Sie, ob die Datei gültig ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets `zip-with-dependencies.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Bei einem gültigen Appium Java JUnit-Paket sollte die Ausgabe wie folgt aussehen:

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
everything built from the ./src/test directory)
```

```
|– zip-with-dependencies.zip (this .zip file contains all of the items)
`– dependency-jars (this is the directory that contains all of your dependencies,
 built as JAR files)
 |– com.some-dependency.bar-4.1.jar
 |– com.another-dependency.thing-1.0.jar
 |– joda-time-2.7.jar
 `– log4j-1.2.14.jar
```

## APPIUM\_WEB\_JAVA\_JUNIT\_TEST\_PACKAGE\_DEPENDENCY\_DIR\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Das Verzeichnis `dependency-jars` konnte in Ihrem Testpaket nicht gefunden werden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob das Verzeichnis `dependency-jars` in dem Paket enthalten ist, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets `zip-with-dependencies.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das Appium Java JUnit-Paket gültig ist, befindet sich das Verzeichnis *dependency-jars* im Arbeitsverzeichnis:

```
.
|– acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
```

```
|– acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
everything built from the ./src/test directory)
|– zip-with-dependencies.zip (this .zip file contains all of the items)
`– dependency-jars (this is the directory that contains all of your dependencies,
built as JAR files)
 |– com.some-dependency.bar-4.1.jar
 |– com.another-dependency.thing-1.0.jar
 |– joda-time-2.7.jar
 `– log4j-1.2.14.jar
```

## APPIUM\_WEB\_JAVA\_JUNIT\_TEST\_PACKAGE\_JAR\_MISSING\_IN\_DEPENDEN

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

In der Verzeichnisstruktur von „dependency-jars“ konnte keine JAR-Datei gefunden werden. Extrahieren Sie Ihr Testpaket und öffnen Sie dann das Verzeichnis „dependency-jars“; überprüfen Sie, ob sich im Verzeichnis mindestens eine JAR-Datei befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets zip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das Appium Java JUnit-Paket gültig ist, befindet sich mindestens eine *.jar*-Datei im Verzeichnis *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

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

In Ihrem Testpaket konnte keine „\*-tests.jar“-Datei gefunden werden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob mindestens eine „\*-tests.jar“ in dem Paket enthalten ist, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets `zip-with-dependencies.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das Appium Java JUnit-Paket gültig ist, finden Sie mindestens eine *Krug*-Datei wie *acme-android-appium-1.0-Snapshot-Tests.jar* in unserem Beispiel. Bei anderen Projekten lautet der Dateiname anders, er sollte jedoch in jedem Fall mit *-tests.jar* enden.

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
 built as JAR files)
 |- com.some-dependency.bar-4.1.jar
 |- com.another-dependency.thing-1.0.jar
 |- joda-time-2.7.jar
 `- log4j-1.2.14.jar
```

## APPIUM\_WEB\_JAVA\_JUNIT\_TEST\_PACKAGE\_CLASS\_FILE\_MISSING\_IN\_TESTS

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

In der JAR-Datei der Tests konnte keine Klassendatei gefunden werden. Extrahieren Sie Ihr ZIP-Testpaket und dekomprimieren Sie dann die JAR-Datei für die Tests; überprüfen Sie, ob sich in der JAR-Datei mindestens eine Klassendatei befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets `zip-with-dependencies.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Du solltest mindestens eine JAR-Datei finden wie *acme-android-appium-1.0-Snapshot-Tests.jar* in unserem Beispiel. Bei anderen Projekten lautet der Dateiname anders, er sollte jedoch in jedem Fall mit *-tests.jar* enden.

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
 built as JAR files)
 |- com.some-dependency.bar-4.1.jar
 |- com.another-dependency.thing-1.0.jar
 |- joda-time-2.7.jar
 `- log4j-1.2.14.jar
```

3. Nachdem Sie die Dateien erfolgreich extrahiert haben, sollte in der Struktur des Arbeitsverzeichnisses bei der Ausführung des folgenden Befehls mindestens eine Klasse angezeigt werden:

```
$ tree .
```

Sie sollten eine Ausgabe wie die Folgende sehen:

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|- one-class-file.class
|- folder
| `- another-class-file.class
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
 built as JAR files)
 |- com.some-dependency.bar-4.1.jar
 |- com.another-dependency.thing-1.0.jar
 |- joda-time-2.7.jar
 `- log4j-1.2.14.jar
```

# APPIUM\_WEB\_JAVA\_JUNIT\_TEST\_PACKAGE\_JUNIT\_VERSION\_VALUE\_UNK

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

## Warning

Es konnte kein Wert für die JUnit-Version gefunden werden. Extrahieren Sie Ihr Testpaket und öffnen Sie das Verzeichnis „dependency-jars“; überprüfen Sie, ob sich im Verzeichnis mindestens eine JUnit-JAR-Datei befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Paketszip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
tree .
```

Die Ausgabe sollte in etwa wie folgt aussehen:

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
 built as JAR files)
 |- junit-4.10.jar
 |- com.some-dependency.bar-4.1.jar
 |- com.another-dependency.thing-1.0.jar
 |- joda-time-2.7.jar
 `- log4j-1.2.14.jar
```

Wenn das Appium Java JUnit-Paket gültig ist, enthält es in unserem Beispiel mindestens eine JUnit-Abhängigkeitsdatei ähnlich der Datei *junit-4.10.jar*. Der Dateiname setzt sich dabei aus dem Schlüsselwort *junit*, gefolgt von der Versionsnummer (in diesem Beispiel 4.10) zusammen.

## APPIUM\_WEB\_JAVA\_JUNIT\_TEST\_PACKAGE\_INVALID\_JUNIT\_VERSION

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Es wurde erkannt, dass JUnit in einer Vorversion von 4.10, der minimalen unterstützten Version, verwendet wird. Bitte ändern Sie die JUnit-Version und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Paketszip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

In dem Verzeichnis sollte eine JUnit-Abhängigkeitsdatei (in unserem Beispiel *junit-4.10.jar*) mit der richtigen Versionsnummer (im Beispiel 4.10) enthalten sein:

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
```

```
`- dependency-jars (this is the directory that contains all of your dependencies,
 built as JAR files)
 |- junit-4.10.jar
 |- com.some-dependency.bar-4.1.jar
 |- com.another-dependency.thing-1.0.jar
 |- joda-time-2.7.jar
 `- log4j-1.2.14.jar
```

### Note

Ihre Tests werden möglicherweise nicht ordnungsgemäß ausgeführt, wenn die für JUnit angegebene Version in Ihrem Testpaket eine Vorversion der minimal unterstützten Version 4.10 ist.

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## Fehlerbehebung bei Appium Java TestNG-Tests in AWS Device Farm

Im folgenden Thema werden Fehlermeldungen aufgeführt, die beim Hochladen von Appium Java TestNG-Tests auftreten können, und Umgehungen für die einzelnen Fehler empfohlen.

### Note

Die folgenden Anweisungen gelten für Linux x86\_64 and Mac.

## APPIUM\_JAVA\_TESTNG\_TEST\_PACKAGE\_UNZIP\_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Ihre ZIP-Datei für den Test konnte nicht geöffnet werden. Prüfen Sie, ob die Datei gültig ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets `zip-with-dependencies.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Bei einem gültigen Appium Java JUnit-Paket sollte die Ausgabe wie folgt aussehen:

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
 built as JAR files)
 |- com.some-dependency.bar-4.1.jar
 |- com.another-dependency.thing-1.0.jar
 |- joda-time-2.7.jar
 `- log4j-1.2.14.jar
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_JAVA\_TESTNG\_TEST\_PACKAGE\_DEPENDENCY\_DIR\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

**⚠ Warning**

Das Verzeichnis `dependency-jars` konnte in Ihrem Testpaket nicht gefunden werden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob das Verzeichnis `dependency-jars` im Paket enthalten ist, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets `zip-with-dependencies.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das Appium Java JUnit-Paket gültig ist, befindet sich das Verzeichnis *dependency-jars* im Arbeitsverzeichnis.

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
 built as JAR files)
 |- com.some-dependency.bar-4.1.jar
 |- com.another-dependency.thing-1.0.jar
 |- joda-time-2.7.jar
 `- log4j-1.2.14.jar
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_JAVA\_TESTNG\_TEST\_PACKAGE\_JAR\_MISSING\_IN\_DEPENDENCY\_DIR

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

In der Verzeichnisstruktur von „dependency-jars“ konnte keine JAR-Datei gefunden werden. Extrahieren Sie Ihr Testpaket und öffnen Sie dann das Verzeichnis „dependency-jars“; überprüfen Sie, ob sich im Verzeichnis mindestens eine JAR-Datei befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Paketszip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das Appium Java JUnit-Paket gültig ist, befindet sich mindestens eine *.jar*-Datei im Verzeichnis *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
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_JAVA\_TESTNG\_TEST\_PACKAGE\_TESTS\_JAR\_FILE\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

In Ihrem Testpaket konnte keine „\*-tests.jar“-Datei gefunden werden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob mindestens eine „\*-tests.jar“ in dem Paket enthalten ist, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Paketszip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das Appium Java JUnit-Paket gültig ist, finden Sie mindestens eine *Krug* Datei wie *acme-android-appium-1.0-Snapshot-Tests.jar* in unserem Beispiel. Bei anderen Projekten lautet der Dateiname anders, er sollte jedoch in jedem Fall mit *-tests.jar* enden.

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
 built as JAR files)
 |- com.some-dependency.bar-4.1.jar
```

```
|– com.another-dependency.thing-1.0.jar
|– joda-time-2.7.jar
`– log4j-1.2.14.jar
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_JAVA\_TESTNG\_TEST\_PACKAGE\_CLASS\_FILE\_MISSING\_IN\_TESTS

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

In der JAR-Datei der Tests konnte keine Klassendatei gefunden werden. Extrahieren Sie Ihr ZIP-Testpaket und dekomprimieren Sie dann die JAR-Datei für die Tests; überprüfen Sie, ob sich in der JAR-Datei mindestens eine Klassendatei befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Paketszip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Du solltest mindestens eine JAR-Datei finden wie *acme-android-appium-1.0-Snapshot-Tests.jar* in unserem Beispiel. Bei anderen Projekten lautet der Dateiname anders, er sollte jedoch in jedem Fall mit *-tests.jar* enden.

```
.
|– acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
```

```
|– acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
everything built from the ./src/test directory)
|– zip-with-dependencies.zip (this .zip file contains all of the items)
`– dependency-jars (this is the directory that contains all of your dependencies,
built as JAR files)
 |– com.some-dependency.bar-4.1.jar
 |– com.another-dependency.thing-1.0.jar
 |– joda-time-2.7.jar
 `– log4j-1.2.14.jar
```

3. Um Dateien aus der jar-Datei zu extrahieren, können Sie den folgenden Befehl ausführen:

```
$ jar xf acme-android-appium-1.0-SNAPSHOT-tests.jar
```

4. Nachdem Sie die Dateien erfolgreich extrahiert haben, führen Sie den folgenden Befehl aus:

```
$ tree .
```

Sie sollten mindestens eine Klasse in der Arbeitsverzeichnisstruktur finden:

```
.
|– acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
built from the ./src/main directory)
|– acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
everything built from the ./src/test directory)
|– one-class-file.class
|– folder
| `– another-class-file.class
|– zip-with-dependencies.zip (this .zip file contains all of the items)
`– dependency-jars (this is the directory that contains all of your dependencies,
built as JAR files)
 |– com.some-dependency.bar-4.1.jar
 |– com.another-dependency.thing-1.0.jar
 |– joda-time-2.7.jar
 `– log4j-1.2.14.jar
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

# Fehlerbehebung bei Appium Java TestNG-Webanwendungen in AWS Device Farm

Im folgenden Thema werden Fehlermeldungen aufgelistet, die beim Hochladen von Appium Java TestNG-Tests für Webanwendungen auftreten können, und Behelfslösungen für die einzelnen Fehler empfohlen.

## APPIUM\_WEB\_JAVA\_TESTNG\_TEST\_PACKAGE\_UNZIP\_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Ihre ZIP-Datei für den Test konnte nicht geöffnet werden. Prüfen Sie, ob die Datei gültig ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets `zip-with-dependencies.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Bei einem gültigen Appium Java JUnit-Paket sollte die Ausgabe wie folgt aussehen:

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
```

```
`- dependency-jars (this is the directory that contains all of your dependencies,
 built as JAR files)
 |- com.some-dependency.bar-4.1.jar
 |- com.another-dependency.thing-1.0.jar
 |- joda-time-2.7.jar
 `- log4j-1.2.14.jar
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_WEB\_JAVA\_TESTNG\_TEST\_PACKAGE\_DEPENDENCY\_DIR\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Das Verzeichnis `dependency-jars` konnte in Ihrem Testpaket nicht gefunden werden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob das Verzeichnis `dependency-jars` in dem Paket enthalten ist, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets `zip-with-dependencies.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das Appium Java JUnit-Paket gültig ist, befindet sich das Verzeichnis *dependency-jars* im Arbeitsverzeichnis.

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
```

```
|– acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
everything built from the ./src/test directory)
|– zip-with-dependencies.zip (this .zip file contains all of the items)
`– dependency-jars (this is the directory that contains all of your dependencies,
built as JAR files)
 |– com.some-dependency.bar-4.1.jar
 |– com.another-dependency.thing-1.0.jar
 |– joda-time-2.7.jar
 `– log4j-1.2.14.jar
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_WEB\_JAVA\_TESTNG\_TEST\_PACKAGE\_JAR\_MISSING\_IN\_DEPENDENCY\_DIR

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

In der Verzeichnisstruktur von „*dependency-jars*“ konnte keine JAR-Datei gefunden werden. Extrahieren Sie Ihr Testpaket und öffnen Sie dann das Verzeichnis „*dependency-jars*“; überprüfen Sie, ob sich im Verzeichnis mindestens eine JAR-Datei befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets `zip-with-dependencies.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das Appium Java JUnit-Paket gültig ist, befindet sich mindestens eine *.jar*-Datei im Verzeichnis *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
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_WEB\_JAVA\_TESTNG\_TEST\_PACKAGE\_TESTS\_JAR\_FILE\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

In Ihrem Testpaket konnte keine „\*-tests.jar“-Datei gefunden werden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob mindestens eine „\*-tests.jar“ in dem Paket enthalten ist, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Paketszip-with-dependencies.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das Appium Java JUnit-Paket gültig ist, finden Sie mindestens eine *Krug*-Datei wie *acme-android-appium-1.0-Snapshot-Tests.jar* in unserem Beispiel. Bei anderen Projekten lautet der Dateiname anders, er sollte jedoch in jedem Fall mit *-tests.jar* enden.

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
 built as JAR files)
 |- com.some-dependency.bar-4.1.jar
 |- com.another-dependency.thing-1.0.jar
 |- joda-time-2.7.jar
 `- log4j-1.2.14.jar
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_WEB\_JAVA\_TESTNG\_TEST\_PACKAGE\_CLASS\_FILE\_MISSING\_IN\_TESTS\_JAR

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

In der JAR-Datei der Tests konnte keine Klassendatei gefunden werden. Extrahieren Sie Ihr ZIP-Testpaket und dekomprimieren Sie dann die JAR-Datei für die Tests; überprüfen Sie, ob sich in der JAR-Datei mindestens eine Klassendatei befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets `zip-with-dependencies.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip zip-with-dependencies.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Du solltest mindestens eine JAR-Datei finden wie *acme-android-appium-1.0-Snapshot-Tests.jar* in unserem Beispiel. Bei anderen Projekten lautet der Dateiname anders, er sollte jedoch in jedem Fall mit *-tests.jar* enden.

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
 built as JAR files)
 |- com.some-dependency.bar-4.1.jar
 |- com.another-dependency.thing-1.0.jar
 |- joda-time-2.7.jar
 `- log4j-1.2.14.jar
```

3. Um Dateien aus der jar-Datei zu extrahieren, können Sie den folgenden Befehl ausführen:

```
$ jar xf acme-android-appium-1.0-SNAPSHOT-tests.jar
```

4. Nachdem Sie die Dateien erfolgreich extrahiert haben, führen Sie den folgenden Befehl aus:

```
$ tree .
```

Sie sollten mindestens eine Klasse in der Arbeitsverzeichnisstruktur finden:

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|- one-class-file.class
|- folder
| `- another-class-file.class
|- zip-with-dependencies.zip (this .zip file contains all of the items)
```

```
`- dependency-jars (this is the directory that contains all of your dependencies,
 built as JAR files)
 |- com.some-dependency.bar-4.1.jar
 |- com.another-dependency.thing-1.0.jar
 |- joda-time-2.7.jar
 `- log4j-1.2.14.jar
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## Fehlerbehebung bei Appium-Python-Tests in AWS Device Farm

Im folgenden Thema werden Fehlermeldungen aufgeführt, die beim Hochladen von Appium Python-Tests auftreten können, und Behelfslösungen für die einzelnen Fehler empfohlen.

### APPIUM\_PYTHON\_TEST\_PACKAGE\_UNZIP\_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

#### Warning

Wir konnten Ihre ZIP-Datei für den Appium-Test nicht öffnen. Prüfen Sie, ob die Datei gültig ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets `test_bundle.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip test_bundle.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Bei einem gültigen Appium Python-Paket sollte die Ausgabe wie folgt aussehen:

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
| |-- test_unittest.py
|-- wheelhouse (directory)
 |-- Appium_Python_Client-0.20-cp27-none-any.whl
 |-- py-1.4.31-py2.py3-none-any.whl
 |-- pytest-2.9.0-py2.py3-none-any.whl
 |-- selenium-2.52.0-cp27-none-any.whl
 |-- wheel-0.26.0-py2.py3-none-any.whl
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_PYTHON\_TEST\_PACKAGE\_DEPENDENCY\_WHEEL\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Wir konnten keine Wheel-Datei für die Abhängigkeiten in der Wheelhouse-Verzeichnisstruktur finden. Extrahieren Sie Ihr Testpaket und öffnen Sie dann das Verzeichnis „wheelhouse“; überprüfen Sie, ob sich im Verzeichnis mindestens eine Wheel-Datei befindet, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets `test_bundle.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip test_bundle.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das Appium Python-Paket gültig ist, befindet sich mindestens eine abhängige *.whl*-Datei im Verzeichnis *wheelhouse* (wie die hervorgehobenen Dateien im folgenden Beispiel).

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
| |-- test_unittest.py
|-- wheelhouse (directory)
| |-- Appium_Python_Client-0.20-cp27-none-any.whl
| |-- py-1.4.31-py2.py3-none-any.whl
| |-- pytest-2.9.0-py2.py3-none-any.whl
| |-- selenium-2.52.0-cp27-none-any.whl
| |-- wheel-0.26.0-py2.py3-none-any.whl
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_PYTHON\_TEST\_PACKAGE\_INVALID\_PLATFORM

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Wir haben festgestellt, dass in mindestens einer Wheel-Datei eine Plattform angegeben ist, die nicht unterstützt wird. Extrahieren Sie Ihr Testpaket und öffnen Sie dann das Verzeichnis „wheelhouse“; überprüfen Sie, ob die Namen der Wheel-Dateien auf *-any.whl* oder *-linux\_x86\_64.whl* enden, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets *test\_bundle.zip*.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip test_bundle.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das Appium Python-Paket gültig ist, befindet sich mindestens eine abhängige *.whl*-Datei im Verzeichnis *wheelhouse* (wie die hervorgehobenen Dateien im folgenden Beispiel). Die Dateinamen können variieren, sollten jedoch in jedem Fall auf *-any.whl* oder *-linux\_x86\_64.whl* enden; diese Zeichenfolgen bezeichnen die jeweilige Plattform. Andere Plattformen wie windows werden nicht unterstützt.

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
| |-- test_unittest.py
|-- wheelhouse (directory)
| |-- Appium_Python_Client-0.20-cp27-none-any.whl
| |-- py-1.4.31-py2.py3-none-any.whl
| |-- pytest-2.9.0-py2.py3-none-any.whl
| |-- selenium-2.52.0-cp27-none-any.whl
| |-- wheel-0.26.0-py2.py3-none-any.whl
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_PYTHON\_TEST\_PACKAGE\_TEST\_DIR\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Wir konnten das Verzeichnis „tests“ nicht in Ihrem Testpaket finden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob das Verzeichnis „tests“ in dem Paket enthalten ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets `test_bundle.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip test_bundle.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das Appium Python-Paket gültig ist, befindet sich das Verzeichnis `tests` im Arbeitsverzeichnis.

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
| |-- test_unittest.py
|-- wheelhouse (directory)
| |-- Appium_Python_Client-0.20-cp27-none-any.whl
| |-- py-1.4.31-py2.py3-none-any.whl
| |-- pytest-2.9.0-py2.py3-none-any.whl
| |-- selenium-2.52.0-cp27-none-any.whl
| |-- wheel-0.26.0-py2.py3-none-any.whl
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_PYTHON\_TEST\_PACKAGE\_INVALID\_TEST\_FILE\_NAME

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Wir konnten keine gültige Test-Datei in der Verzeichnisstruktur für die Tests finden.

Extrahieren Sie Ihr Paket und öffnen Sie dann das Verzeichnis „tests“; überprüfen Sie, ob der

Name mindestens einer Datei mit dem Schlüsselwort „test“ beginnt oder darauf endet, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets `test_bundle.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip test_bundle.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das Appium Python-Paket gültig ist, befindet sich das Verzeichnis `tests` im Arbeitsverzeichnis. Die Dateinamen können variieren, sollten jedoch in jedem Fall mit `test_` beginnen oder auf `_test.py` enden.

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
| |-- test_unittest.py
|-- wheelhouse (directory)
| |-- Appium_Python_Client-0.20-cp27-none-any.whl
| |-- py-1.4.31-py2.py3-none-any.whl
| |-- pytest-2.9.0-py2.py3-none-any.whl
| |-- selenium-2.52.0-cp27-none-any.whl
| |-- wheel-0.26.0-py2.py3-none-any.whl
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_PYTHON\_TEST\_PACKAGE\_REQUIREMENTS\_TXT\_FILE\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

**⚠ Warning**

Wir konnten die Datei „requirements.txt“ nicht in Ihrem Testpaket finden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob die Datei „requirements.txt“ in dem Paket enthalten ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets `test_bundle.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip test_bundle.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das Appium Python-Paket gültig ist, befindet sich die Datei *requirements.txt* im Arbeitsverzeichnis.

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
| |-- test_unittest.py
|-- wheelhouse (directory)
| |-- Appium_Python_Client-0.20-cp27-none-any.whl
| |-- py-1.4.31-py2.py3-none-any.whl
| |-- pytest-2.9.0-py2.py3-none-any.whl
| |-- selenium-2.52.0-cp27-none-any.whl
| |-- wheel-0.26.0-py2.py3-none-any.whl
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_PYTHON\_TEST\_PACKAGE\_INVALID\_PYTEST\_VERSION

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Wir haben festgestellt, dass pytest in einer Version niedriger als 2.8.0 verwendet wird, der minimalen unterstützten Version. Bitte ändern Sie die Version von pytest in der Datei „requirements.txt“ und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets `test_bundle.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip test_bundle.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Sie sollten das finden `requirements.txt` Datei im Arbeitsverzeichnis.

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
| |-- test_unittest.py
|-- wheelhouse (directory)
| |-- Appium_Python_Client-0.20-cp27-none-any.whl
| |-- py-1.4.31-py2.py3-none-any.whl
| |-- pytest-2.9.0-py2.py3-none-any.whl
| |-- selenium-2.52.0-cp27-none-any.whl
| |-- wheel-0.26.0-py2.py3-none-any.whl
```

3. Sie können die Version von pytest abrufen, indem Sie den folgenden Befehl ausführen:

```
$ grep "pytest" requirements.txt
```

Die Ausgabe sollte folgendermaßen aussehen:

```
pytest==2.9.0
```

Die Version von pytest wird angezeigt, in diesem Beispiel 2.9.0. Wenn das Appium Python-Paket gültig ist, sollte pytest in der Version 2.8.0 oder höher verwendet werden.

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_PYTHON\_TEST\_PACKAGE\_INSTALL\_DEPENDENCY\_WHEELS\_FAIL

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Wir konnten die abhängigen Wheel-Dateien nicht installieren. Extrahieren Sie Ihr Testpaket und öffnen Sie die Datei „requirements.txt“ sowie das Verzeichnis „wheelhouse“; überprüfen Sie, ob die in der Datei „requirements.txt“ angegebenen abhängigen Wheel-Dateien mit denen im Verzeichnis „wheelhouse“ übereinstimmen, und versuchen Sie es erneut.

Wir empfehlen dringend, das Modul [Python virtualenv](#) für die Überprüfung von Paketen zu installieren. Das folgende Beispiel zeigt, wie Sie eine virtuelle Umgebung mit Python virtualenv erstellen und diese anschließend aktivieren:

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

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets test\_bundle.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip test_bundle.zip
```

2. Sie können die Installation von Wheel-Dateien testen, indem Sie den folgenden Befehl ausführen:

```
$ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./requirements.txt
```

Bei einem gültigen Appium Python-Paket sollte die Ausgabe wie folgt aussehen:

```
Ignoring indexes: https://pypi.python.org/simple
Collecting Appium-Python-Client==0.20 (from -r ./requirements.txt (line 1))
Collecting py==1.4.31 (from -r ./requirements.txt (line 2))
Collecting pytest==2.9.0 (from -r ./requirements.txt (line 3))
Collecting selenium==2.52.0 (from -r ./requirements.txt (line 4))
Collecting wheel==0.26.0 (from -r ./requirements.txt (line 5))
Installing collected packages: selenium, Appium-Python-Client, py, pytest, wheel
 Found existing installation: wheel 0.29.0
 Uninstalling wheel-0.29.0:
 Successfully uninstalled wheel-0.29.0
Successfully installed Appium-Python-Client-0.20 py-1.4.31 pytest-2.9.0
selenium-2.52.0 wheel-0.26.0
```

3. Sie können die virtuelle Umgebung deaktivieren, indem Sie den folgenden Befehl ausführen:

```
$ deactivate
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_PYTHON\_TEST\_PACKAGE\_PYTEST\_COLLECT\_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Wir konnten im Verzeichnis „tests“ keine Tests erfassen. Extrahieren Sie Ihr Testpaket, überprüfen Sie mit dem Befehl `py.test --collect-only <path to your tests`

`directory>`, ob das Testpaket gültig ist, und versuchen Sie es erneut, wenn der Befehl keine Fehler zurückgibt.

Wir empfehlen dringend, das Modul [Python virtualenv](#) für die Überprüfung von Paketen zu installieren. Das folgende Beispiel zeigt, wie Sie eine virtuelle Umgebung mit Python virtualenv erstellen und diese anschließend aktivieren:

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

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets `test_bundle.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip test_bundle.zip
```

2. Sie können die Wheel-Dateien installieren, indem Sie den folgenden Befehl ausführen:

```
$ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./requirements.txt
```

3. Sie können die Tests erfassen, indem Sie den folgenden Befehl ausführen:

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

Bei einem gültigen Appium Python-Paket sollte die Ausgabe wie folgt aussehen:

```
===== test session starts =====
platform darwin -- Python 2.7.11, pytest-2.9.0, py-1.4.31, pluggy-0.3.1
rootdir: /Users/zhen/Desktop/Ios/tests, inifile:
collected 1 items
<Module 'test_unittest.py'>
 <UnitTestCase 'DeviceFarmAppiumWebTests'>
 <TestCaseFunction 'test_devicefarm'>

===== no tests ran in 0.11 seconds =====
```

4. Sie können die virtuelle Umgebung deaktivieren, indem Sie den folgenden Befehl ausführen:

```
$ deactivate
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_PYTHON\_TEST\_PACKAGE\_DEPENDENCY\_WHEELS\_INSUFFICIENT

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Wir konnten im Wheelhouse-Verzeichnis nicht genügend Radabhängigkeiten finden. Bitte entpacken Sie Ihr Testpaket und öffnen Sie dann das Wheelhouse-Verzeichnis. Stellen Sie sicher, dass Sie alle Radabhängigkeiten in der Datei `requirements.txt` angegeben haben.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets `test_bundle.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip test_bundle.zip
```

2. Überprüfen Sie die Länge der `requirements.txt`-Datei sowie die Anzahl der `.whl`-abhängige Dateien im Wheelhouse-Verzeichnis:

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

Wenn die Anzahl von `.whl`-abhängigen Dateien kleiner ist als die Anzahl der nicht leeren Zeilen in Ihrer `requirements.txt`-Datei, dann müssen Sie Folgendes sicherstellen:

- Es gibt eine `.whl`-abhängige Datei, die jeder Zeile in der `requirements.txt`-Datei entspricht.

- Es gibt keine anderen Zeilen in der `requirements.txt`-Datei, die andere Informationen als die Namen der Abhängigkeitspakete enthält.
- In der Datei werden keine Abhängigkeitsnamen in mehreren Zeilen dupliziert `requirements.txt`-Datei, sodass zwei Zeilen in der Datei einer entsprechen können. `.whl`-abhängige Datei.

AWS Device Farm unterstützt keine Zeilen in `requirements.txt`-Dateien, die nicht direkt Abhängigkeitspaketen entsprechen, wie z. B. Zeilen, die globale Optionen für die Angabe von `pip install`-Befehl. Siehe [Dateiformat für Anforderungen](#) für eine Liste globaler Optionen.

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## Fehlerbehebung bei Appium-Python-Webanwendungstests in AWS Device Farm

Im folgenden Thema werden Fehlermeldungen aufgelistet, die beim Hochladen von Appium Python for Web Application-Tests auftreten können, und Umgehungen für die einzelnen Fehler empfohlen.

### APPIUM\_WEB\_PYTHON\_TEST\_PACKAGE\_UNZIP\_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

#### Warning

Wir konnten Ihre ZIP-Datei für den Appium-Test nicht öffnen. Prüfen Sie, ob die Datei gültig ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets `test_bundle.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip test_bundle.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Bei einem gültigen Appium Python-Paket sollte die Ausgabe wie folgt aussehen:

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
| |-- test_unittest.py
|-- wheelhouse (directory)
| |-- Appium_Python_Client-0.20-cp27-none-any.whl
| |-- py-1.4.31-py2.py3-none-any.whl
| |-- pytest-2.9.0-py2.py3-none-any.whl
| |-- selenium-2.52.0-cp27-none-any.whl
| |-- wheel-0.26.0-py2.py3-none-any.whl
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_WEB\_PYTHON\_TEST\_PACKAGE\_DEPENDENCY\_WHEEL\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Wir konnten keine Wheel-Datei für die Abhängigkeiten in der Wheelhouse-Verzeichnisstruktur finden. Extrahieren Sie Ihr Testpaket und öffnen Sie dann das Verzeichnis „wheelhouse“; überprüfen Sie, ob sich im Verzeichnis mindestens eine Wheel-Datei befindet, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets `test_bundle.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip test_bundle.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das Appium Python-Paket gültig ist, befindet sich mindestens eine abhängige *.whl*-Datei im Verzeichnis *wheelhouse* (wie die hervorgehobenen Dateien im folgenden Beispiel).

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
| |-- test_unittest.py
|-- wheelhouse (directory)
| |-- Appium_Python_Client-0.20-cp27-none-any.whl
| |-- py-1.4.31-py2.py3-none-any.whl
| |-- pytest-2.9.0-py2.py3-none-any.whl
| |-- selenium-2.52.0-cp27-none-any.whl
| |-- wheel-0.26.0-py2.py3-none-any.whl
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_WEB\_PYTHON\_TEST\_PACKAGE\_INVALID\_PLATFORM

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Wir haben festgestellt, dass in mindestens einer Wheel-Datei eine Plattform angegeben ist, die nicht unterstützt wird. Extrahieren Sie Ihr Testpaket und öffnen Sie dann das Verzeichnis „wheelhouse“; überprüfen Sie, ob die Namen der Wheel-Dateien auf *-any.whl* oder *-linux\_x86\_64.whl* enden, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets `test_bundle.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip test_bundle.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das Appium Python-Paket gültig ist, befindet sich mindestens eine abhängige `.whl`-Datei im Verzeichnis `wheelhouse` (wie die hervorgehobenen Dateien im folgenden Beispiel). Die Dateinamen können variieren, sollten jedoch in jedem Fall auf `-any.whl` oder `-linux_x86_64.whl` enden; diese Zeichenfolgen bezeichnen die jeweilige Plattform. Andere Plattformen wie `windows` werden nicht unterstützt.

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
| |-- test_unittest.py
|-- wheelhouse (directory)
| |-- Appium_Python_Client-0.20-cp27-none-any.whl
| |-- py-1.4.31-py2.py3-none-any.whl
| |-- pytest-2.9.0-py2.py3-none-any.whl
| |-- selenium-2.52.0-cp27-none-any.whl
| |-- wheel-0.26.0-py2.py3-none-any.whl
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_WEB\_PYTHON\_TEST\_PACKAGE\_TEST\_DIR\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

**⚠ Warning**

Wir konnten das Verzeichnis „tests“ nicht in Ihrem Testpaket finden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob das Verzeichnis „tests“ in dem Paket enthalten ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets `test_bundle.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip test_bundle.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das Appium Python-Paket gültig ist, befindet sich das Verzeichnis `tests` im Arbeitsverzeichnis.

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
| |-- test_unittest.py
|-- wheelhouse (directory)
| |-- Appium_Python_Client-0.20-cp27-none-any.whl
| |-- py-1.4.31-py2.py3-none-any.whl
| |-- pytest-2.9.0-py2.py3-none-any.whl
| |-- selenium-2.52.0-cp27-none-any.whl
| |-- wheel-0.26.0-py2.py3-none-any.whl
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_WEB\_PYTHON\_TEST\_PACKAGE\_INVALID\_TEST\_FILE\_NAME

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Wir konnten keine gültige Test-Datei in der Verzeichnisstruktur für die Tests finden. Extrahieren Sie Ihr Paket und öffnen Sie dann das Verzeichnis „tests“; überprüfen Sie, ob der Name mindestens einer Datei mit dem Schlüsselwort „test“ beginnt oder darauf endet, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets `test_bundle.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip test_bundle.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das Appium Python-Paket gültig ist, befindet sich das Verzeichnis `tests` im Arbeitsverzeichnis. Die Dateinamen können variieren, sollten jedoch in jedem Fall mit `test_` beginnen oder auf `_test.py` enden.

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
| |-- test_unittest.py
|-- wheelhouse (directory)
| |-- Appium_Python_Client-0.20-cp27-none-any.whl
| |-- py-1.4.31-py2.py3-none-any.whl
| |-- pytest-2.9.0-py2.py3-none-any.whl
| |-- selenium-2.52.0-cp27-none-any.whl
```

```
`-- wheel-0.26.0-py2.py3-none-any.whl
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_WEB\_PYTHON\_TEST\_PACKAGE\_REQUIREMENTS\_TXT\_FILE\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Wir konnten die Datei „requirements.txt“ nicht in Ihrem Testpaket finden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob die Datei „requirements.txt“ in dem Paket enthalten ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets test\_bundle.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip test_bundle.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das Appium Python-Paket gültig ist, befindet sich die Datei *requirements.txt* im Arbeitsverzeichnis.

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
| |-- test_unittest.py
|-- wheelhouse (directory)
 |-- Appium_Python_Client-0.20-cp27-none-any.whl
```

```
|-- py-1.4.31-py2.py3-none-any.whl
|-- pytest-2.9.0-py2.py3-none-any.whl
|-- selenium-2.52.0-cp27-none-any.whl
`-- wheel-0.26.0-py2.py3-none-any.whl
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_WEB\_PYTHON\_TEST\_PACKAGE\_INVALID\_PYTEST\_VERSION

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Wir haben festgestellt, dass pytest in einer Version niedriger als 2.8.0 verwendet wird, der minimalen unterstützten Version. Bitte ändern Sie die Version von pytest in der Datei „requirements.txt“ und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets test\_bundle.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip test_bundle.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Die Datei *requirements.txt* sollte sich im Arbeitsverzeichnis befinden.

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
| |-- test_unittest.py
```

```
`-- wheelhouse (directory)
 |-- Appium_Python_Client-0.20-cp27-none-any.whl
 |-- py-1.4.31-py2.py3-none-any.whl
 |-- pytest-2.9.0-py2.py3-none-any.whl
 |-- selenium-2.52.0-cp27-none-any.whl
 `-- wheel-0.26.0-py2.py3-none-any.whl
```

3. Sie können die Version von pytest abrufen, indem Sie den folgenden Befehl ausführen:

```
$ grep "pytest" requirements.txt
```

Die Ausgabe sollte folgendermaßen aussehen:

```
pytest==2.9.0
```

Die Version von pytest wird angezeigt, in diesem Beispiel 2.9.0. Wenn das Appium Python-Paket gültig ist, sollte pytest in der Version 2.8.0 oder höher verwendet werden.

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_WEB\_PYTHON\_TEST\_PACKAGE\_INSTALL\_DEPENDENCY\_WHEELS\_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Wir konnten die abhängigen Wheel-Dateien nicht installieren. Extrahieren Sie Ihr Testpaket und öffnen Sie die Datei „requirements.txt“ sowie das Verzeichnis „wheelhouse“; überprüfen Sie, ob die in der Datei „requirements.txt“ angegebenen abhängigen Wheel-Dateien mit denen im Verzeichnis „wheelhouse“ übereinstimmen, und versuchen Sie es erneut.

Wir empfehlen dringend, das Modul [Python virtualenv](#) für die Überprüfung von Paketen zu installieren. Das folgende Beispiel zeigt, wie Sie eine virtuelle Umgebung mit Python virtualenv erstellen und diese anschließend aktivieren:

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

```
$ source bin/activate
```

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets `test_bundle.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip test_bundle.zip
```

2. Sie können die Installation von Wheel-Dateien testen, indem Sie den folgenden Befehl ausführen:

```
$ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./requirements.txt
```

Bei einem gültigen Appium Python-Paket sollte die Ausgabe wie folgt aussehen:

```
Ignoring indexes: https://pypi.python.org/simple
Collecting Appium-Python-Client==0.20 (from -r ./requirements.txt (line 1))
Collecting py==1.4.31 (from -r ./requirements.txt (line 2))
Collecting pytest==2.9.0 (from -r ./requirements.txt (line 3))
Collecting selenium==2.52.0 (from -r ./requirements.txt (line 4))
Collecting wheel==0.26.0 (from -r ./requirements.txt (line 5))
Installing collected packages: selenium, Appium-Python-Client, py, pytest, wheel
 Found existing installation: wheel 0.29.0
 Uninstalling wheel-0.29.0:
 Successfully uninstalled wheel-0.29.0
Successfully installed Appium-Python-Client-0.20 py-1.4.31 pytest-2.9.0
selenium-2.52.0 wheel-0.26.0
```

3. Sie können die virtuelle Umgebung deaktivieren, indem Sie den folgenden Befehl ausführen:

```
$ deactivate
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## APPIUM\_WEB\_PYTHON\_TEST\_PACKAGE\_PYTEST\_COLLECT\_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Wir konnten im Verzeichnis „tests“ keine Tests erfassen. Extrahieren Sie Ihr Testpaket, überprüfen Sie mit dem Befehl "py.test --collect-only <Pfad zu Ihrem Testverzeichnis>", ob das Testpaket gültig ist, und versuchen Sie es erneut, wenn der Befehl keine Fehler zurückgibt.

Wir empfehlen dringend, das Modul [Python virtualenv](#) für die Überprüfung von Paketen zu installieren. Das folgende Beispiel zeigt, wie Sie eine virtuelle Umgebung mit Python virtualenv erstellen und diese anschließend aktivieren:

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

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets test\_bundle.zip.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip test_bundle.zip
```

2. Sie können die Wheel-Dateien installieren, indem Sie den folgenden Befehl ausführen:

```
$ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./requirements.txt
```

3. Sie können die Tests erfassen, indem Sie den folgenden Befehl ausführen:

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

Bei einem gültigen Appium Python-Paket sollte die Ausgabe wie folgt aussehen:

```
===== test session starts =====
platform darwin -- Python 2.7.11, pytest-2.9.0, py-1.4.31, pluggy-0.3.1
rootdir: /Users/zhenal/Desktop/Ios/tests, inifile:
collected 1 items
<Module 'test_unittest.py'>
 <UnitTestCase 'DeviceFarmAppiumWebTests'>
 <TestCaseFunction 'test_devicefarm'>

===== no tests ran in 0.11 seconds =====
```

4. Sie können die virtuelle Umgebung deaktivieren, indem Sie den folgenden Befehl ausführen:

```
$ deactivate
```

Weitere Informationen finden Sie unter [Arbeiten mit Appium und AWS Device Farm](#).

## Fehlerbehebung bei Instrumentierungstests in AWS Device Farm

Im folgenden Thema werden Fehlermeldungen aufgeführt, die beim Hochladen von Instrumentierungstests auftreten können, und Umgehungen für die einzelnen Fehler empfohlen.

### INSTRUMENTATION\_TEST\_PACKAGE\_UNZIP\_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

#### Warning

Ihre APK-Datei für den Test konnte nicht geöffnet werden. Prüfen Sie, ob die Datei gültig ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Paket für den Test fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets `app-debug-androidTest-unaligned.apk`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip app-debug-androidTest-unaligned.apk
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Bei einem gültigen Instrumentierungstestpaket sieht die Ausgabe wie folgt aus:

```
.
|-- AndroidManifest.xml
|-- classes.dex
|-- resources.arsc
|-- LICENSE-junit.txt
|-- junit (directory)
`-- META-INF (directory)
```

Weitere Informationen finden Sie unter [Arbeiten mit Instrumentierung für Android und AWS Device Farm](#).

## INSTRUMENTATION\_TEST\_PACKAGE\_AAPT\_DEBUG\_BADGING\_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Wir konnten keine Informationen zu Ihrem Testpaket extrahieren. Überprüfen Sie mit dem Befehl "aapt debug badging <Pfad zu Ihrem Testpaket>", ob das Testpaket gültig ist, und versuchen Sie es erneut, wenn der Befehl keine Fehler zurückgibt.

Während des Upload-Validierungsprozesses analysiert Device Farm Informationen aus der Ausgabe von `aapt debug badging <path to your package>` Befehl.

Stellen Sie sicher, dass Sie diesen Befehl erfolgreich für Ihr Instrumentierungstestpaket ausführen können.

Im folgenden Beispiel lautet der Name des Paketsapp-debug-androidTest-unaligned.apk.

- Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ aapt debug badging app-debug-androidTest-unaligned.apk
```

Bei einem gültigen Instrumentierungstestpaket sieht die Ausgabe wie folgt aus:

```
package: name='com.amazon.aws.adf.android.referenceapp.test' versionCode=''
 versionName='' platformBuildVersionName='5.1.1-1819727'
sdkVersion:'9'
targetSdkVersion:'22'
application-label:'Test-api'
application: label='Test-api' icon=''
application-debuggable
uses-library:'android.test.runner'
feature-group: label=''
uses-feature: name='android.hardware.touchscreen'
uses-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'
```

Weitere Informationen finden Sie unter [Arbeiten mit Instrumentierung für Android und AWS Device Farm](#).

## INSTRUMENTATION\_TEST\_PACKAGE\_INSTRUMENTATION\_RUNNER\_VALU

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Wir konnten den Wert für Instrumentation Runner nicht in derAndroidManifest.xml. Bitte überprüfen Sie, ob das Testpaket gültig ist, indem Sie den Befehl „aapt dump xmltree“ ausführen <path to your test package>AndroidManifest.xml“, und versuchen Sie es erneut,

nachdem Sie den Instrumentierungs-Runner-Wert hinter dem Schlüsselwort „Instrumentation“ gefunden haben.

Während der Upload-Validierung analysiert Device Farm den Instrumentation Runner-Wert aus dem XML-Analysebaum für eine XML-Datei, die im Paket enthalten ist. Sie können folgenden Befehl verwenden: `aapt dump xmltree <path to your package> AndroidManifest.xml`.

Stellen Sie sicher, dass Sie diesen Befehl für Ihr Instrumentierungstestpaket ausführen und den Instrumentierungswert erfolgreich finden können.

Im folgenden Beispiel lautet der Name des Pakets `app-debug-androidTest-unaligned.apk`.

- Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ aapt dump xmltree app-debug-androidTest-unaligned.apk AndroidManifest.xml | grep -A5 "instrumentation"
```

Bei einem gültigen Instrumentierungstestpaket sieht die Ausgabe wie folgt aus:

```
E: instrumentation (line=9)
 A: android:label(0x01010001)="Tests for
com.amazon.aws.adf.android.referenceapp" (Raw: "Tests for
com.amazon.aws.adf.android.referenceapp")
 A:
android:name(0x01010003)="android.support.test.runner.AndroidJUnitRunner" (Raw:
"android.support.test.runner.AndroidJUnitRunner")
 A:
android:targetPackage(0x01010021)="com.amazon.aws.adf.android.referenceapp" (Raw:
"com.amazon.aws.adf.android.referenceapp")
 A: android:handleProfiling(0x01010022)=(type 0x12)0x0
 A: android:functionalTest(0x01010023)=(type 0x12)0x0
```

Weitere Informationen finden Sie unter [Arbeiten mit Instrumentierung für Android und AWS Device Farm](#).

## INSTRUMENTATION\_TEST\_PACKAGE\_AAPT\_DUMP\_XMLTREE\_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Wir konnten das gültige nicht finden `AndroidManifest.xml` in Ihrem Testpaket. Bitte überprüfen Sie, ob das Testpaket gültig ist, indem Sie den Befehl „`aapt dump xmltree`“ ausführen `<path to your test package>AndroidManifest.xml`“, und versuchen Sie es erneut, wenn der Befehl keinen Fehler ausgibt.

Während der Upload-Validierung analysiert Device Farm mithilfe des folgenden Befehls Informationen aus der XML-Analysestruktur nach einer im Paket enthaltenen XML-Datei: `aapt dump xmltree <path to your package> AndroidManifest.xml`.

Stellen Sie sicher, dass Sie diesen Befehl erfolgreich für Ihr Instrumentierungstestpaket ausführen können.

Im folgenden Beispiel lautet der Name des Pakets `app-debug-androidTest-unaligned.apk`.

- Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ aapt dump xmltree app-debug-androidTest-unaligned.apk AndroidManifest.xml
```

Bei einem gültigen Instrumentierungstestpaket sieht die Ausgabe wie folgt aus:

```
N: android=http://schemas.android.com/apk/res/android
E: manifest (line=2)
 A: package="com.amazon.aws.adf.android.referenceapp.test" (Raw:
"com.amazon.aws.adf.android.referenceapp.test")
 A: platformBuildVersionCode=(type 0x10)0x16 (Raw: "22")
 A: platformBuildVersionName="5.1.1-1819727" (Raw: "5.1.1-1819727")
E: uses-sdk (line=5)
 A: android:minSdkVersion(0x0101020c)=(type 0x10)0x9
 A: android:targetSdkVersion(0x01010270)=(type 0x10)0x16
E: instrumentation (line=9)
```

```

A: android:label(0x01010001)="Tests for
com.amazon.aws.adf.android.referenceapp" (Raw: "Tests for
com.amazon.aws.adf.android.referenceapp")
A:
android:name(0x01010003)="android.support.test.runner.AndroidJUnitRunner" (Raw:
"android.support.test.runner.AndroidJUnitRunner")
A:
android:targetPackage(0x01010021)="com.amazon.aws.adf.android.referenceapp" (Raw:
"com.amazon.aws.adf.android.referenceapp")
A: android:handleProfiling(0x01010022)=(type 0x12)0x0
A: android:functionalTest(0x01010023)=(type 0x12)0x0
E: application (line=16)
A: android:label(0x01010001)=@0x7f020000
A: android:debuggable(0x0101000f)=(type 0x12)0xffffffff
E: uses-library (line=17)
A: android:name(0x01010003)="android.test.runner" (Raw:
"android.test.runner")

```

Weitere Informationen finden Sie unter [Arbeiten mit Instrumentierung für Android und AWS Device Farm](#).

## INSTRUMENTATION\_TEST\_PACKAGE\_TEST\_PACKAGE\_NAME\_VALUE\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Der Paketname konnte in Ihrem Testpaket nicht gefunden werden. Überprüfen Sie, ob das Testpaket gültig ist, indem Sie den Befehl "aapt debug badging <Pfad zu Ihrem Testpaket>" ausführen, und wiederholen Sie den Vorgang, nachdem Sie den Wert für den Paketnamen gefunden haben, der dem Schlüsselwort "package: name" folgt.

Während der Upload-Validierung analysiert Device Farm den Wert des Paketnamens aus der Ausgabe des folgenden Befehls: aapt debug badging <path to your package>.

Stellen Sie sicher, dass Sie diesen Befehl für Ihr Instrumentierungstestpaket ausführen und den Wert für den Paketnamen erfolgreich finden können.

Im folgenden Beispiel lautet der Name des Pakets app-debug-androidTest-unaligned.apk.

- Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ aapt debug badging app-debug-androidTest-unaligned.apk | grep "package: name="
```

Bei einem gültigen Instrumentierungstestpaket sieht die Ausgabe wie folgt aus:

```
package: name='com.amazon.aws.adf.android.referenceapp.test' versionCode=''
versionName='' platformBuildVersionName='5.1.1-1819727'
```

Weitere Informationen finden Sie unter [Arbeiten mit Instrumentierung für Android und AWS Device Farm](#).

## Fehlerbehebung bei iOS-Anwendungstests in AWS Device Farm

Im folgenden Thema werden Fehlermeldungen aufgeführt, die beim Hochladen von iOS-Anwendungstests auftreten können, und Umgehungen für die einzelnen Fehler empfohlen.

### Note

Die folgenden Anweisungen gelten für Linux x86\_64 and Mac.

## IOS\_APP\_UNZIP\_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Ihre Anwendung konnte nicht geöffnet werden. Prüfen Sie, ob die Datei gültig ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Anwendungspaket fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets `AWSDeviceFarmiOSReferenceApp.ipa`.

1. Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Bei einem gültigen iOS-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

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

Weitere Informationen finden Sie unter [Arbeiten mit iOS-Tests in AWS Device Farm](#).

## IOS\_APP\_PAYLOAD\_DIR\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Wir konnten das Nutzlastverzeichnis nicht in Ihrer Anwendung finden. Extrahieren Sie Ihre Anwendung, überprüfen Sie, ob das Nutzlastverzeichnis im Paket enthalten ist, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets `AWSDeviceFarmiOSReferenceApp.ipa`.

1. Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das iOS-Anwendungspaket gültig ist, befindet sich das Verzeichnis für die *Nutzlast* im Arbeitsverzeichnis.

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

Weitere Informationen finden Sie unter [Arbeiten mit iOS-Tests in AWS Device Farm](#).

## IOS\_APP\_APP\_DIR\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Das .app-Verzeichnis konnte im Nutzlastverzeichnis nicht gefunden werden. Extrahieren Sie Ihre Anwendung und öffnen Sie dann das Nutzlastverzeichnis. Überprüfen Sie, ob sich das .app-Verzeichnis im Verzeichnis befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets `AWSDeviceFarmiOSReferenceApp.ipa`.

1. Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das iOS-Anwendungspaket gültig ist, finden Sie eine *.app*-Verzeichnis wie *AWSDeviceFarmiOSReferenceApp.app* in unserem Beispiel in der *NutzLast*-Verzeichnis.

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

Weitere Informationen finden Sie unter [Arbeiten mit iOS-Tests in AWS Device Farm](#).

## IOS\_APP\_PLIST\_FILE\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Die Datei Info.plist konnte im *.app*-Verzeichnis nicht gefunden werden. Extrahieren Sie Ihre Anwendung und öffnen Sie dann das *.app*-Verzeichnis. Überprüfen Sie, ob sich die Datei Info.plist im Verzeichnis befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets *AWSDeviceFarmiOSReferenceApp.ipa*.

1. Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das iOS-Anwendungspaket gültig ist, finden Sie *Info.plist* Datei im *.app* Verzeichnis wie *AWSDeviceFarmiOSReferenceApp.app* in unserem Beispiel.

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

Weitere Informationen finden Sie unter [Arbeiten mit iOS-Tests in AWS Device Farm](#).

## IOS\_APP\_CPU\_ARCHITECTURE\_VALUE\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Der Wert für die CPU-Architektur konnte in der Datei *Info.plist* nicht gefunden werden. Bitte entpacken Sie Ihre Anwendung und öffnen Sie dann die Datei *Info.plist* im Verzeichnis *.app*. Vergewissern Sie sich, dass der Schlüssel „*UIRequiredDeviceCapabilities*“ ist angegeben, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets *AWSDeviceFarmiOSReferenceApp.ipa*.

1. Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Du solltest die finden *Info.plist* Datei in einem *.app* Verzeichnis wie *AWSDeviceFarmiOSReferenceApp.app* in unserem Beispiel:

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

- Um den Wert für die CPU-Architektur zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

```
$ pip install biplist
```

- Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['UIRequiredDeviceCapabilities']
```

Bei einem gültigen iOS-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

```
['armv7']
```

Weitere Informationen finden Sie unter [Arbeiten mit iOS-Tests in AWS Device Farm](#).

## IOS\_APP\_PLATFORM\_VALUE\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Der Wert für die Plattform konnte in der Datei Info.plist nicht gefunden werden. Bitte entpacken Sie Ihre Anwendung und öffnen Sie dann die Datei Info.plist im Verzeichnis .app. Vergewissern Sie sich, dass der Schlüssel „CFBundleSupportedPlatforms“ ist angegeben, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets `AWSDDeviceFarmiOSReferenceApp.ipa`.

1. Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip AWSDDeviceFarmiOSReferenceApp.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Sie sollten das finden `Info.plist` Datei in einem `.app` Verzeichnis wie `AWSDDeviceFarmiOSReferenceApp.app` in unserem Beispiel:

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

3. Um den Wert für die Plattform zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

Für Python können Sie das `biplist`-Modul installieren, indem Sie den folgenden Befehl ausführen:

```
$ pip install biplist
```

4. Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDDeviceFarmiOSReferenceApp-cal.app/Info.plist')
print info_plist['CFBundleSupportedPlatforms']
```

Bei einem gültigen iOS-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

```
['iPhoneOS']
```

Weitere Informationen finden Sie unter [Arbeiten mit iOS-Tests in AWS Device Farm](#).

## IOS\_APP\_WRONG\_PLATFORM\_DEVICE\_VALUE

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Wir haben festgestellt, dass der Wert für das Plattformgerät in der Datei Info.plist falsch war. Bitte entpacken Sie Ihre Anwendung und öffnen Sie dann die Datei Info.plist im Verzeichnis .app. Überprüfen Sie, ob der Wert des Schlüssels „CFBundleSupportedPlatforms“ enthält nicht das Schlüsselwort „Simulator“, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets `AWSDeviceFarmiOSReferenceApp.ipa`.

1. Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Du solltest die finden `Info.plist` Datei in einem `.app` Verzeichnis wie `AWSDeviceFarmiOSReferenceApp.app` in unserem Beispiel:

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

3. Um den Wert für die Plattform zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

Für Python können Sie das `Bplist`-Modul installieren, indem Sie den folgenden Befehl ausführen:

```
$ pip install biplist
```

- Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['CFBundleSupportedPlatforms']
```

Bei einem gültigen iOS-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

```
['iPhoneOS']
```

Damit die iOS-Anwendung gültig ist, darf der Wert nicht das Schlüsselwort `simulator` enthalten.

Weitere Informationen finden Sie unter [Arbeiten mit iOS-Tests in AWS Device Farm](#).

## IOS\_APP\_FORM\_FACTOR\_VALUE\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Der Wert für den Formfaktor konnte in der Datei Info.plist nicht gefunden werden. Bitte entpacken Sie Ihre Anwendung und öffnen Sie dann die Datei Info.plist im Verzeichnis `.app`. Vergewissern Sie sich, dass der Schlüssel „UIDeviceFamily“ ist angegeben, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets `AWSDeviceFarmiOSReferenceApp.ipa`.

- Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

- Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Du solltest die finden *Info.plist* Datei in einem *.app* Verzeichnis wie *AWSDeviceFarmiOSReferenceApp.app* in unserem Beispiel:

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

- Um den Wert für den Formfaktor zu ermitteln, können Sie mithilfe von Xcode oder Python *Info.plist* öffnen.

Für Python können Sie das *biplist*-Modul installieren, indem Sie den folgenden Befehl ausführen:

```
$ pip install biplist
```

- Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/Info.plist')
print info_plist['UIDeviceFamily']
```

Bei einem gültigen iOS-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

```
[1, 2]
```

Weitere Informationen finden Sie unter [Arbeiten mit iOS-Tests in AWS Device Farm](#).

## IOS\_APP\_PACKAGE\_NAME\_VALUE\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

**⚠ Warning**

Der Wert für den Paketnamen konnte in der Datei Info.plist nicht gefunden werden. Bitte entpacken Sie Ihre Anwendung und öffnen Sie dann die Datei Info.plist im Verzeichnis .app. Vergewissern Sie sich, dass der Schlüssel „CFBundleIdentifier“ angegeben ist, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets `AWSDeviceFarmiOSReferenceApp.ipa`.

1. Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Du solltest das finden `Info.plist` Datei in einem `.app` Verzeichnis wie `AWSDeviceFarmiOSReferenceApp.app` in unserem Beispiel:

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

3. Um den Wert für den Paketnamen zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

```
$ pip install biplist
```

4. Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

```
import biplist
```

```
info_plist = bplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/Info.plist')
print info_plist['CFBundleIdentifier']
```

Bei einem gültigen iOS-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

```
Amazon.AWSDeviceFarmiOSReferenceApp
```

Weitere Informationen finden Sie unter [Arbeiten mit iOS-Tests in AWS Device Farm](#).

## IOS\_APP\_EXECUTABLE\_VALUE\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Der Wert für die ausführbare Datei konnte in der Datei Info.plist nicht gefunden werden. Bitte entpacken Sie Ihre Anwendung und öffnen Sie dann die Datei Info.plist im Verzeichnis .app. Vergewissern Sie sich, dass der Schlüssel „CFBundleExecutable“ angegeben, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets `AWSDeviceFarmiOSReferenceApp.ipa`.

1. Kopieren Sie das Anwendungspaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Du solltest die finden *Info.plist* Datei in einem *.app* Verzeichnis wie *AWSDeviceFarmiOSReferenceApp.app* in unserem Beispiel:

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

3. Um den Wert für die ausführbare Datei zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

```
$ pip install biplist
```

4. Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['CFBundleExecutable']
```

Bei einem gültigen iOS-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

```
AWSDeviceFarmiOSReferenceApp
```

Weitere Informationen finden Sie unter [Arbeiten mit iOS-Tests in AWS Device Farm](#).

## Fehlerbehebung bei XCTest-Tests in AWS Device Farm

Im folgenden Thema werden Fehlermeldungen aufgeführt, die beim Hochladen von XCTest-Tests auftreten können, und Umgehungen für die einzelnen Fehler empfohlen.

### Note

Bei den folgenden Anweisungen wird davon ausgegangen, dass Sie MacOS verwenden.

## XCTEST\_TEST\_PACKAGE\_UNZIP\_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Ihre ZIP-Datei für den Test konnte nicht geöffnet werden. Prüfen Sie, ob die Datei gültig ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Anwendungspaket fehlerfrei dekomprimieren können. Im folgenden Beispiel lautet der Name des Pakets `swiftExampleTests.xctest-1.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Bei einem gültigen XCTest-Paket sollte die Ausgabe wie folgt aussehen:

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

Weitere Informationen finden Sie unter [Arbeiten mit XCTest für iOS und AWS Device Farm](#).

## XCTEST\_TEST\_PACKAGE\_XCTEST\_DIR\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

**⚠ Warning**

Wir konnten das `.xctest`-Verzeichnis nicht in Ihrem Testpaket finden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob das `.xctest`-Verzeichnis in dem Paket enthalten ist, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets `swiftExampleTests.xctest-1.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das XCTest-Paket gültig ist, finden Sie ein Verzeichnis mit einem ähnlichen Namen `swiftExampleTests.xctest` innerhalb des Arbeitsverzeichnisses. Der Name sollte mit `.xctest` enden.

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

Weitere Informationen finden Sie unter [Arbeiten mit XCTest für iOS und AWS Device Farm](#).

## XCTEST\_TEST\_PACKAGE\_PLIST\_FILE\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

**⚠ Warning**

Die Datei `Info.plist` konnte im `.xctest`-Verzeichnis nicht gefunden werden. Extrahieren Sie Ihr Testpaket und öffnen Sie dann das `.xctest`-Verzeichnis. Überprüfen Sie, ob sich die Datei `Info.plist` im Verzeichnis befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel lautet der Name des Pakets `swiftExampleTests.xctest-1.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das XCTest-Paket gültig ist, befindet sich die Datei `Info.plist` im `.xctest`-Verzeichnis. In unserem Beispiel unten heißt das Verzeichnis `swiftExampleTests.xctest`.

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

Weitere Informationen finden Sie unter [Arbeiten mit XCTest für iOS und AWS Device Farm](#).

## XCTEST\_TEST\_PACKAGE\_PACKAGE\_NAME\_VALUE\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

**⚠ Warning**

Der Wert für den Paketnamen konnte in der Datei Info.plist nicht gefunden werden. Bitte entpacken Sie Ihr Testpaket und öffnen Sie dann die Datei Info.plist. Vergewissern Sie sich, dass der Schlüssel „CFBundleIdentifier“ angegeben ist, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets `swiftExampleTests.xctest-1.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Du solltest das finden *Info.plist* Datei in einem *.xctest* Verzeichnis wie *swiftExampleTests.xctest* in unserem Beispiel:

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

3. Um den Wert für den Paketnamen zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

```
$ pip install biplist
```

4. Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

```
import biplist
info_plist = biplist.readPlist('swiftExampleTests.xctest/Info.plist')
print info_plist['CFBundleIdentifier']
```

Bei einem gültigen XCTest-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

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

Weitere Informationen finden Sie unter [Arbeiten mit XCTest für iOS und AWS Device Farm](#).

## XCTEST\_TEST\_PACKAGE\_EXECUTABLE\_VALUE\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Der Wert für die ausführbare Datei konnte in der Datei Info.plist nicht gefunden werden. Bitte entpacken Sie Ihr Testpaket und öffnen Sie dann die Datei Info.plist. Vergewissern Sie sich, dass der Schlüssel „CFBundleExecutable“ angegeben ist, und versuchen Sie es erneut.

Im folgenden Beispiel lautet der Name des Pakets `swiftExampleTests.xctest-1.zip`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Du solltest das finden *Info.plist* Datei in einem *.xctest* Verzeichnis wie *swiftExampleTests.xctest* in unserem Beispiel:

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

- Um den Wert für den Paketnamen zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

```
$ pip install biplist
```

- Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

```
import biplist
info_plist = biplist.readPlist('swiftExampleTests.xctest/Info.plist')
print info_plist['CFBundleExecutable']
```

Bei einem gültigen XCTest-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

```
swiftExampleTests
```

Weitere Informationen finden Sie unter [Arbeiten mit XCTest für iOS und AWS Device Farm](#).

## Fehlerbehebung bei XCTest-UI-Tests in AWS Device Farm

Im folgenden Thema werden Fehlermeldungen aufgeführt, die beim Hochladen von XCTest UI-Tests auftreten können, und Umgehungen für die einzelnen Fehler empfohlen.

### Note

Die folgenden Anweisungen gelten für Linux x86\_64 and Mac.

## XCTEST\_UI\_TEST\_PACKAGE\_UNZIP\_FAILED

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Ihre IPA-Datei für den Test konnte nicht geöffnet werden. Prüfen Sie, ob die Datei gültig ist, und versuchen Sie es erneut.

Stellen Sie sicher, dass Sie das Anwendungspaket fehlerfrei dekomprimieren können. Im folgenden Beispiel ist der Name des Pakets `swift-sample-UI.ipa`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip swift-sample-UI.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Bei einem gültigen iOS-Anwendungspaket sollte die Ausgabe wie folgt aussehen:

```
.
|-- Payload (directory)
 |-- swift-sampleUITests-Runner.app (directory)
 |-- Info.plist
 |-- Plugins (directory)
 |-- `swift-sampleUITests.xctest (directory)
 |-- Info.plist
 |-- `-- (any other files)
 |-- `-- (any other files)
```

Weitere Informationen finden Sie unter [XCTest-UI](#).

## XCTEST\_UI\_TEST\_PACKAGE\_PAYLOAD\_DIR\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Wir konnten das Nutzlastverzeichnis nicht in Ihrem Testpaket finden. Extrahieren Sie Ihr Testpaket, überprüfen Sie, ob das Nutzlastverzeichnis im Paket enthalten ist, und versuchen Sie es erneut.

Im folgenden Beispiel ist der Name des Pakets `swift-sample-UI.ipa`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip swift-sample-UI.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das XCTest-UI-Paket gültig ist, befindet sich das Verzeichnis für die *Nutzlast* im Arbeitsverzeichnis.

```
.
|-- Payload (directory)
 |-- swift-sampleUITests-Runner.app (directory)
 |-- Info.plist
 |-- Plugins (directory)
 |-- swift-sampleUITests.xctest (directory)
 |-- Info.plist
 |-- (any other files)
 |-- (any other files)
```

Weitere Informationen finden Sie unter [XCTest-UI](#).

## XCTEST\_UI\_TEST\_PACKAGE\_APP\_DIR\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Das `.app`-Verzeichnis konnte im Nutzlastverzeichnis nicht gefunden werden. Extrahieren Sie Ihr Testpaket und öffnen Sie dann das Nutzlastverzeichnis. Überprüfen Sie, ob sich das `.app`-Verzeichnis im Verzeichnis befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel ist der Name des Pakets `swift-sample-UI.ipa`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip swift-sample-UI.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das XCTest-UI-Paket gültig ist, finden Sie ein `.app`-Verzeichnis wie `swift-sampleUITests-Runner.app` im Beispiel innerhalb des Verzeichnisses für die `Nutzlast`.

```
.
|-- Payload (directory)
 |-- swift-sampleUITests-Runner.app (directory)
 |-- Info.plist
 |-- Plugins (directory)
 |-- swift-sampleUITests.xctest (directory)
 |-- Info.plist
 |-- (any other files)
 |-- (any other files)
```

Weitere Informationen finden Sie unter [XCTest-UI](#).

## XCTEST\_UI\_TEST\_PACKAGE\_PLUGINS\_DIR\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Das Plug-In-Verzeichnis konnte im `.app`-Verzeichnis nicht gefunden werden. Extrahieren Sie Ihr Testpaket und öffnen Sie dann das `.app`-Verzeichnis. Überprüfen Sie, ob sich das Plug-In-Verzeichnis im Verzeichnis befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel ist der Name des Pakets `swift-sample-UI.ipa`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip swift-sample-UI.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das XCTest-UI-Paket gültig ist, befindet sich das Verzeichnis *Plug-Ins* in einem *.app*-Verzeichnis. In diesem Beispiel ist der Name des Verzeichnisses *swift-sampleUITests-Runner.app*.

```
.
|-- Payload (directory)
 |-- swift-sampleUITests-Runner.app (directory)
 |-- Info.plist
 |-- Plugins (directory)
 |-- swift-sampleUITests.xctest (directory)
 |-- Info.plist
 |-- (any other files)
 |-- (any other files)
```

Weitere Informationen finden Sie unter [XCTest-UI](#).

## XCTEST\_UI\_TEST\_PACKAGE\_XCTEST\_DIR\_MISSING\_IN\_PLUGINS\_DIR

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Das `.xctest`-Verzeichnis konnte im Plug-In-Verzeichnis nicht gefunden werden. Extrahieren Sie Ihr Testpaket und öffnen Sie dann das Plug-In-Verzeichnis. Überprüfen Sie, ob sich das `.xctest`-Verzeichnis im Verzeichnis befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel ist der Name des Pakets `swift-sample-UI.ipa`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip swift-sample-UI.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das XCTest-UI-Paket gültig ist, befindet sich das Verzeichnis `.xctest` im *Plug-In*-Verzeichnis. In diesem Beispiel ist der Name des Verzeichnisses `swift-sampleUITests.xctest`.

```
.
|-- Payload (directory)
 |-- swift-sampleUITests-Runner.app (directory)
 |-- Info.plist
 |-- Plugins (directory)
 |-- swift-sampleUITests.xctest (directory)
 |-- Info.plist
 |-- (any other files)
 |-- (any other files)
```

Weitere Informationen finden Sie unter [XCTest-UI](#).

## XCTEST\_UI\_TEST\_PACKAGE\_PLIST\_FILE\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Die Datei `Info.plist` konnte im `.app`-Verzeichnis nicht gefunden werden. Extrahieren Sie Ihr Testpaket und öffnen Sie dann das `.app`-Verzeichnis. Überprüfen Sie, ob sich die Datei `Info.plist` im Verzeichnis befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel ist der Name des Pakets `swift-sample-UI.ipa`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip swift-sample-UI.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das XCTest-UI-Paket gültig ist, befindet sich die Datei *Info.plist* im *.app*-Verzeichnis. Im folgenden Beispiel ist der Name des Verzeichnisses *swift-sampleUITests-Runner.app*.

```
.
|-- Payload (directory)
 |-- swift-sampleUITests-Runner.app (directory)
 |-- Info.plist
 |-- Plugins (directory)
 |-- swift-sampleUITests.xctest (directory)
 |-- Info.plist
 |-- (any other files)
 |-- (any other files)
```

Weitere Informationen finden Sie unter [XCTest-UI](#).

## XCTEST\_UI\_TEST\_PACKAGE\_PLIST\_FILE\_MISSING\_IN\_XCTEST\_DIR

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Die Datei `Info.plist` konnte im `.xctest`-Verzeichnis nicht gefunden werden. Extrahieren Sie Ihr Testpaket und öffnen Sie dann das `.xctest`-Verzeichnis. Überprüfen Sie, ob sich die Datei `Info.plist` im Verzeichnis befindet, und wiederholen Sie den Vorgang.

Im folgenden Beispiel ist der Name des Pakets `swift-sample-UI.ipa`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip swift-sample-UI.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Wenn das XCTest-UI-Paket gültig ist, befindet sich die Datei *Info.plist* im *.xctest*-Verzeichnis. Im folgenden Beispiel ist der Name des Verzeichnisses *swift-sampleUITests.xctest*.

```
.
|-- Payload (directory)
 |-- swift-sampleUITests-Runner.app (directory)
 |-- Info.plist
 |-- Plugins (directory)
 |-- swift-sampleUITests.xctest (directory)
 |-- Info.plist
 |-- (any other files)
 |-- (any other files)
```

Weitere Informationen finden Sie unter [XCTest-UI](#).

## XCTEST\_UI\_TEST\_PACKAGE\_CPU\_ARCHITECTURE\_VALUE\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Der Wert für die CPU-Architektur konnte in der Datei `Info.plist` nicht gefunden werden. Bitte entpacken Sie Ihr Testpaket und öffnen Sie dann die Datei `Info.plist` im Verzeichnis `.app`.

Vergewissern Sie sich, dass der Schlüssel „UIRequiredDeviceCapabilities“ angegeben, und versuchen Sie es erneut.

Im folgenden Beispiel ist der Name des Pakets swift-sample-UI.ipa.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip swift-sample-UI.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Sie sollten die Datei *Info.plist* in einem *.app*-Verzeichnis wie *swift-sampleUITests-Runner.app* in unserem Beispiel finden:

```
.
|-- Payload (directory)
 |-- swift-sampleUITests-Runner.app (directory)
 |-- Info.plist
 |-- Plugins (directory)
 |-- swift-sampleUITests.xctest (directory)
 |-- Info.plist
 |-- (any other files)
 |-- (any other files)
```

3. Um den Wert für die CPU-Architektur zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

```
$ pip install biplist
```

4. Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
```

```
print info_plist['UIRequiredDeviceCapabilities']
```

Bei einem gültigen XCTest-UI-Paket sollte die Ausgabe wie folgt aussehen:

```
['armv7']
```

Weitere Informationen finden Sie unter [XCTest-UI](#).

## XCTEST\_UI\_TEST\_PACKAGE\_PLATFORM\_VALUE\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Der Wert für die Plattform konnte in der Datei Info.plist nicht gefunden werden. Bitte entpacken Sie Ihr Testpaket und öffnen Sie dann die Datei Info.plist im Verzeichnis `.app`. Vergewissern Sie sich, dass der Schlüssel „CFBundleSupportedPlatforms“ ist angegeben, und versuchen Sie es erneut.

Im folgenden Beispiel ist der Name des Pakets `swift-sample-UI.ipa`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip swift-sample-UI.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Sie sollten die Datei *Info.plist* in einem `.app`-Verzeichnis wie *swift-sampleUITests-Runner.app* in unserem Beispiel finden:

```
.
|-- Payload (directory)
```

```
`-- swift-sampleUITests-Runner.app (directory)
 |-- Info.plist
 |-- Plugins (directory)
 | `-- swift-sampleUITests.xctest (directory)
 | |-- Info.plist
 | |-- (any other files)
 `-- (any other files)
```

- Um den Wert für die Plattform zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

```
$ pip install biplist
```

- Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['CFBundleSupportedPlatforms']
```

Bei einem gültigen XCTest-UI-Paket sollte die Ausgabe wie folgt aussehen:

```
['iPhoneOS']
```

Weitere Informationen finden Sie unter [XCTest-UI](#).

## XCTEST\_UI\_TEST\_PACKAGE\_WRONG\_PLATFORM\_DEVICE\_VALUE

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Wir haben festgestellt, dass der Wert für das Plattformgerät in der Datei Info.plist falsch war. Bitte entpacken Sie Ihr Testpaket und öffnen Sie dann die Datei Info.plist im Verzeichnis .app. Überprüfen Sie, ob der Wert des Schlüssels „CFBundleSupportedPlatforms“ enthält nicht das Schlüsselwort „Simulator“, und versuchen Sie es erneut.

Im folgenden Beispiel ist der Name des Pakets `swift-sample-UI.ipa`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip swift-sample-UI.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Sie sollten die Datei *Info.plist* in einem *.app*-Verzeichnis wie *swift-sampleUITests-Runner.app* in unserem Beispiel finden:

```
.
|-- Payload (directory)
 |-- swift-sampleUITests-Runner.app (directory)
 |-- Info.plist
 |-- Plugins (directory)
 |-- swift-sampleUITests.xctest (directory)
 |-- Info.plist
 |-- (any other files)
 |-- (any other files)
```

3. Um den Wert für die Plattform zu ermitteln, können Sie mithilfe von Xcode oder Python `Info.plist` öffnen.

Für Python können Sie das `biplist`-Modul installieren, indem Sie den folgenden Befehl ausführen:

```
$ pip install biplist
```

4. Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['CFBundleSupportedPlatforms']
```

Bei einem gültigen XCtest-UI-Paket sollte die Ausgabe wie folgt aussehen:

```
['iPhoneOS']
```

Damit das XCTest-UI-Paket gültig ist, darf der Wert nicht das Schlüsselwort `simulator` enthalten.

Weitere Informationen finden Sie unter [XCTest-UI](#).

## XCTEST\_UI\_TEST\_PACKAGE\_FORM\_FACTOR\_VALUE\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Der Wert für den Formfaktor konnte in der Datei `Info.plist` nicht gefunden werden. Bitte entpacken Sie Ihr Testpaket und öffnen Sie dann die Datei `Info.plist` im Verzeichnis `.app`. Vergewissern Sie sich, dass der Schlüssel „UI“DeviceFamily“ist angegeben, und versuchen Sie es erneut.

Im folgenden Beispiel ist der Name des Pakets `swift-sample-UI.ipa`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip swift-sample-UI.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Sie sollten die Datei *Info.plist* in einem `.app`-Verzeichnis wie *swift-sampleUITests-Runner.app* in unserem Beispiel finden:

```
.
|-- Payload (directory)
```

```
`-- swift-sampleUITests-Runner.app (directory)
 |-- Info.plist
 |-- Plugins (directory)
 | `-- swift-sampleUITests.xctest (directory)
 | |-- Info.plist
 | |-- (any other files)
 `-- (any other files)
```

- Um den Wert für den Formfaktor zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

```
$ pip install biplist
```

- Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['UIDeviceFamily']
```

Bei einem gültigen XCTest-UI-Paket sollte die Ausgabe wie folgt aussehen:

```
[1, 2]
```

Weitere Informationen finden Sie unter [XCTest-UI](#).

## XCTEST\_UI\_TEST\_PACKAGE\_PACKAGE\_NAME\_VALUE\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Der Wert für den Paketnamen konnte in der Datei Info.plist nicht gefunden werden. Bitte entpacken Sie Ihr Testpaket und öffnen Sie dann die Datei Info.plist im Verzeichnis .app. Vergewissern Sie sich, dass der Schlüssel „CFBundleIdentifier“ angegeben ist, und versuchen Sie es erneut.

Im folgenden Beispiel ist der Name des Pakets `swift-sample-UI.ipa`.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip swift-sample-UI.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Sie sollten die Datei *Info.plist* in einem *.app*-Verzeichnis wie *swift-sampleUITests-Runner.app* in unserem Beispiel finden:

```
.
|-- Payload (directory)
 |-- swift-sampleUITests-Runner.app (directory)
 |-- Info.plist
 |-- Plugins (directory)
 |-- swift-sampleUITests.xctest (directory)
 |-- Info.plist
 |-- (any other files)
 |-- (any other files)
```

3. Um den Wert für den Paketnamen zu ermitteln, können Sie mithilfe von Xcode oder Python `Info.plist` öffnen.

Für Python können Sie das `biplist`-Modul installieren, indem Sie den folgenden Befehl ausführen:

```
$ pip install biplist
```

4. Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['CFBundleIdentifier']
```

Bei einem gültigen XCtest-UI-Paket sollte die Ausgabe wie folgt aussehen:

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

Weitere Informationen finden Sie unter [XCTest-UI](#).

## XCTEST\_UI\_TEST\_PACKAGE\_EXECUTABLE\_VALUE\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Der Wert für die ausführbare Datei konnte in der Datei Info.plist nicht gefunden werden. Bitte entpacken Sie Ihr Testpaket und öffnen Sie dann die Datei Info.plist im Verzeichnis .app. Vergewissern Sie sich, dass der Schlüssel „CFBundleExecutable“ angegeben, und versuchen Sie es erneut.

Im folgenden Beispiel ist der Name des Pakets swift-sample-UI.ipa.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip swift-sample-UI.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Sie sollten die Datei *Info.plist* in einem *.app*-Verzeichnis wie *swift-sampleUITests-Runner.app* in unserem Beispiel finden:

```
.
|-- Payload (directory)
 |-- swift-sampleUITests-Runner.app (directory)
 |-- Info.plist
 |-- Plugins (directory)
 | `-- swift-sampleUITests.xctest (directory)
```

```
|
|
|-- Info.plist
|-- (any other files)
|-- (any other files)
```

- Um den Wert für die ausführbare Datei zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

```
$ pip install biplist
```

- Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['CFBundleExecutable']
```

Bei einem gültigen XCtest-UI-Paket sollte die Ausgabe wie folgt aussehen:

```
XCTRunner
```

Weitere Informationen finden Sie unter [XCtest-UI](#).

## XCTEST\_UI\_TEST\_PACKAGE\_TEST\_PACKAGE\_NAME\_VALUE\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Der Wert für den Paketnamen konnte in der Datei Info.plist im .xctest-Verzeichnis nicht gefunden werden. Bitte entpacken Sie Ihr Testpaket und öffnen Sie dann die Datei Info.plist im Verzeichnis .xctest. Vergewissern Sie sich, dass der Schlüssel „CFBundleIdentifier“ ist angegeben, und versuchen Sie es erneut.

Im folgenden Beispiel ist der Name des Pakets swift-sample-UI.ipa.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip swift-sample-UI.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Sie sollten die Datei *Info.plist* in einem *.app*-Verzeichnis wie *swift-sampleUITests-Runner.app* in unserem Beispiel finden:

```
.
|-- Payload (directory)
 |-- swift-sampleUITests-Runner.app (directory)
 |-- Info.plist
 |-- Plugins (directory)
 |-- swift-sampleUITests.xctest (directory)
 |-- Info.plist
 |-- (any other files)
 |-- (any other files)
```

3. Um den Wert für den Paketnamen zu ermitteln, können Sie mithilfe von Xcode oder Python *Info.plist* öffnen.

Für Python können Sie das *biplist*-Modul installieren, indem Sie den folgenden Befehl ausführen:

```
$ pip install biplist
```

4. Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Plugins/
swift-sampleUITests.xctest/Info.plist')
print info_plist['CFBundleIdentifier']
```

Bei einem gültigen *XCtest-UI*-Paket sollte die Ausgabe wie folgt aussehen:

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

Weitere Informationen finden Sie unter [XCCTest-UI](#).

## XCCTest\_UI\_TEST\_PACKAGE\_TEST\_EXECUTABLE\_VALUE\_MISSING

Wenn die folgende Meldung angezeigt wird, führen Sie die folgenden Schritte aus, um das Problem zu beheben.

### Warning

Der Wert für die ausführbare Datei konnte in der Datei Info.plist im .xcctest-Verzeichnis nicht gefunden werden. Bitte entpacken Sie Ihr Testpaket und öffnen Sie dann die Datei Info.plist im Verzeichnis .xcctest. Vergewissern Sie sich, dass der Schlüssel „CFBundleExecutable“ ist angegeben, und versuchen Sie es erneut.

Im folgenden Beispiel ist der Name des Pakets swift-sample-UI.ipa.

1. Kopieren Sie das Testpaket in Ihr Arbeitsverzeichnis und führen Sie dann den folgenden Befehl aus:

```
$ unzip swift-sample-UI.ipa
```

2. Nachdem Sie das Paket erfolgreich extrahiert haben, können Sie die Baumstruktur für das Arbeitsverzeichnis anzeigen, indem Sie den folgenden Befehl ausführen:

```
$ tree .
```

Sie sollten die Datei *Info.plist* in einem *.app*-Verzeichnis wie *swift-sampleUITests-Runner.app* in unserem Beispiel finden:

```
.
|-- Payload (directory)
 |-- swift-sampleUITests-Runner.app (directory)
 |-- Info.plist
 |-- Plugins (directory)
 |-- `swift-sampleUITests.xcctest (directory)
 |-- Info.plist
 |-- (any other files)
```

```
`-- (any other files)
```

- Um den Wert für die ausführbare Datei zu ermitteln, können Sie mithilfe von Xcode oder Python Info.plist öffnen.

Für Python können Sie das Biplist-Modul installieren, indem Sie den folgenden Befehl ausführen:

```
$ pip install biplist
```

- Öffnen Sie anschließend Python und führen Sie den folgenden Befehl aus:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Plugins/
swift-sampleUITests.xctest/Info.plist')
print info_plist['CFBundleExecutable']
```

Bei einem gültigen XCtest-UI-Paket sollte die Ausgabe wie folgt aussehen:

```
swift-sampleUITests
```

Weitere Informationen finden Sie unter [XCtest-UI](#).

# Sicherheit in AWS Device Farm

Die Sicherheit in der Cloud hat bei AWS höchste Priorität. Als AWS-Kunde profitieren Sie von einer Rechenzentrums- und Netzwerkarchitektur, die zur Erfüllung der Anforderungen von Organisationen entwickelt wurden, für die Sicherheit eine kritische Bedeutung hat.

Sicherheit ist eine übergreifende Verantwortlichkeit zwischen AWS und Ihnen. Das [Modell der geteilten Verantwortung](#) beschreibt dies als Sicherheit der Cloud selbst und Sicherheit in der Cloud:

- Sicherheit der Cloud selbst – AWS ist dafür verantwortlich, die Infrastruktur zu schützen, mit der AWS-Services in der AWS Cloud ausgeführt werden. AWS stellt Ihnen außerdem Services bereit, die Sie sicher nutzen können. Auditoren von Drittanbietern testen und überprüfen die Effektivität unserer Sicherheitsmaßnahmen im Rahmen der [AWS-Compliance-Programme](#) regelmäßig. Informationen zu den Compliance-Programmen, die für AWS Device Farm gelten, finden Sie unter [Betroffene AWS-Services nach Compliance-Programm](#).
- Sicherheit in der Cloud – Ihr Verantwortungsumfang wird durch den AWS-Service bestimmt, den Sie verwenden. Sie sind auch für andere Faktoren verantwortlich, etwa für die Vertraulichkeit Ihrer Daten, für die Anforderungen Ihres Unternehmens und für die geltenden Gesetze und Vorschriften.

Diese Dokumentation hilft Ihnen zu verstehen, wie Sie das Modell der gemeinsamen Verantwortung bei der Verwendung von Device Farm anwenden können. In den folgenden Themen erfahren Sie, wie Sie Device Farm so konfigurieren, dass Sie Ihre Sicherheits- und Compliance-Ziele erreichen. Sie erfahren auch, wie Sie andere AWS-Services nutzen können, die Sie bei der Überwachung und Sicherung Ihrer Device Farm-Ressourcen unterstützen.

## Themen

- [Identity and Access Management in AWS Device Farm](#)
- [Compliance-Validierung für AWS Device Farm](#)
- [Datenschutz in AWS Device Farm](#)
- [Ausfallsicherheit in AWS Device Farm](#)
- [Sicherheit der Infrastruktur in AWS Device Farm](#)
- [Analyse und Verwaltung von Konfigurationsschwachstellen in Device Farm](#)
- [Reaktion auf Vorfälle in Device Farm](#)
- [Protokollierung und Überwachung in Device Farm](#)

- [Bewährte Sicherheitsmethoden für Device Farm](#)

# Identity and Access Management in AWS Device Farm

## Zielgruppe

Wie Sie AWS Identity and Access Management (IAM) verwenden, unterscheidet sich je nach Ihrer Arbeit in Device Farm .

**Service-Benutzer** – Wenn Sie den Device Farm-Service zur Ausführung von Aufgaben verwenden, stellt Ihnen Ihr Administrator die Anmeldeinformationen und Berechtigungen bereit, die Sie benötigen. Wenn Sie für Ihre Arbeit weitere Device Farm-Funktionen ausführen, benötigen Sie möglicherweise zusätzliche Berechtigungen. Wenn Sie die Featuresweise der Zugriffskontrolle nachvollziehen, wissen Sie bereits, welche Berechtigungen Sie von Ihrem Administrator anfordern müssen. Wenn Sie auf ein Feature in Device Farm nicht zugreifen können, finden Sie weitere Informationen unter [Fehlerbehebung für AWS Device Farm-Identität und -Zugriff](#).

**Service-Administrator** – Wenn Sie in Ihrem Unternehmen für Device Farm-Ressourcen verantwortlich sind, haben Sie wahrscheinlich vollständigen Zugriff auf Device Farm . Ihre Aufgabe besteht darin, zu bestimmen, auf welche Device Farm-Funktionen und -Ressourcen Ihre Service-Benutzer zugreifen sollen. Sie müssen dann Anträge an Ihren IAM-Administrator stellen, um die Berechtigungen Ihrer Servicenutzer zu ändern. Lesen Sie die Informationen auf dieser Seite, um die Grundkonzepte von IAM nachzuvollziehen. Weitere Informationen dazu, wie Ihr Unternehmen IAM mit Device Farm verwenden kann, finden Sie unter [Funktionsweise von AWS Device Farm mit IAM](#).

**IAM-Administrator** – Wenn Sie als IAM-Administrator fungieren, sollten Sie Einzelheiten dazu kennen, wie Sie Richtlinien zur Verwaltung des Zugriffs auf Device Farm verfassen können. Beispiele für identitätsbasierte Device Farm-Richtlinien, die Sie in IAM verwenden können, finden Sie unter [Beispiele für identitätsbasierte AWS Device Farm-Richtlinien](#).

## Authentifizierung mit Identitäten

Authentifizierung ist die Art, wie Sie sich mit Ihren Anmeldeinformationen bei AWS anmelden. Die Authentifizierung (Anmeldung bei AWS) muss als Root-Benutzer des AWS-Kontos, als IAM-Benutzer oder durch Übernahme einer IAM-Rolle erfolgen.

Sie können sich bei AWS als Verbundidentität mit Anmeldeinformationen anmelden, die über eine Identitätsquelle bereitgestellt werden. Benutzer von AWS IAM Identity Center (IAM Identity Center),

die Single-Sign-on-Authentifizierung Ihres Unternehmens und Anmeldeinformationen für Google oder Facebook sind Beispiele für Verbundidentitäten. Wenn Sie sich als Verbundidentität anmelden, hat der Administrator vorher mithilfe von IAM-Rollen einen Identitätsverbund eingerichtet. Wenn Sie auf AWS mithilfe des Verbunds zugreifen, übernehmen Sie indirekt eine Rolle.

Je nachdem, welcher Benutzertyp Sie sind, können Sie sich bei der AWS Management Console oder beim AWS-Zugriffportal anmelden. Weitere Informationen zum Anmelden bei AWS finden Sie unter [So melden Sie sich bei Ihrem AWS-Konto an](#) im Benutzerhandbuch von AWS-Anmeldung.

Bei programmgesteuertem Zugriff auf AWS bietet AWS ein Software Development Kit (SDK) und eine Command Line Interface (CLI, Befehlszeilenschnittstelle) zum kryptographischen Signieren Ihrer Anfragen mit Ihren Anmeldeinformationen. Wenn Sie keine AWS-Tools verwenden, müssen Sie Anforderungen selbst signieren. Weitere Informationen zur Verwendung der empfohlenen Methode zum eigenen Signieren von Anforderungen finden Sie unter [Signieren von AWS-API-Anforderungen](#) im IAM-Benutzerhandbuch.

Unabhängig von der verwendeten Authentifizierungsmethode müssen Sie möglicherweise zusätzliche Sicherheitsinformationen angeben. AWS empfiehlt beispielsweise die Verwendung von Multi-Faktor Authentifizierung (MFA), um die Sicherheit Ihres Kontos zu verbessern. Weitere Informationen finden Sie unter [Multi-Faktor-Authentifizierung](#) im AWS IAM Identity Center-Benutzerhandbuch und [Verwenden der Multi-Faktor-Authentifizierung \(MFA\) in AWS](#) im IAM-Benutzerhandbuch.

## AWS-Konto-Root-Benutzer

Wenn Sie ein AWS-Konto neu erstellen, beginnen Sie mit einer Anmeldeidentität, die vollständigen Zugriff auf alle AWS-Services und Ressourcen des Kontos hat. Diese Identität wird als AWS-Konto-Root-Benutzer bezeichnet. Für den Zugriff auf den Root-Benutzer müssen Sie sich mit der E-Mail-Adresse und dem Passwort anmelden, die zur Erstellung des Kontos verwendet wurden. Wir raten ausdrücklich davon ab, den Root-Benutzer für Alltagsaufgaben zu verwenden. Schützen Sie Ihre Root-Benutzer-Anmeldeinformationen und verwenden Sie diese, um die Aufgaben auszuführen, die nur der Root-Benutzer ausführen kann. Eine vollständige Liste der Aufgaben, für die Sie sich als Root-Benutzer anmelden müssen, finden Sie unter [Aufgaben, die Root-Benutzer-Anmeldeinformationen erfordern](#) im IAM-Benutzerhandbuch.

## IAM-Benutzer und -Gruppen

Ein [IAM-Benutzer](#) ist eine Identität in Ihrem AWS-Konto mit bestimmten Berechtigungen für eine einzelne Person oder eine einzelne Anwendung. Wenn möglich, empfehlen wir,

temporäre Anmeldeinformationen zu verwenden, anstatt IAM-Benutzer zu erstellen, die langfristige Anmeldeinformationen wie Passwörter und Zugriffsschlüssel haben. Bei speziellen Anwendungsfällen, die langfristige Anmeldeinformationen mit IAM-Benutzern erfordern, empfehlen wir jedoch, die Zugriffsschlüssel zu rotieren. Weitere Informationen finden Sie unter [Regelmäßiges Rotieren von Zugriffsschlüsseln für Anwendungsfälle, die langfristige Anmeldeinformationen erfordern](#) im IAM-Benutzerhandbuch.

Eine [IAM-Gruppe](#) ist eine Identität, die eine Sammlung von IAM-Benutzern angibt. Sie können sich nicht als Gruppe anmelden. Mithilfe von Gruppen können Sie Berechtigungen für mehrere Benutzer gleichzeitig angeben. Gruppen vereinfachen die Verwaltung von Berechtigungen, wenn es zahlreiche Benutzer gibt. Sie könnten beispielsweise einer Gruppe mit dem Namen IAMAdmins Berechtigungen zum Verwalten von IAM-Ressourcen erteilen.

Benutzer unterscheiden sich von Rollen. Ein Benutzer ist einer einzigen Person oder Anwendung eindeutig zugeordnet. Eine Rolle kann von allen Personen angenommen werden, die sie benötigen. Benutzer besitzen dauerhafte Anmeldeinformationen. Rollen stellen temporäre Anmeldeinformationen bereit. Weitere Informationen finden Sie unter [Erstellen eines IAM-Benutzers \(anstatt einer Rolle\)](#) im IAM-Benutzerhandbuch.

## IAM-Rollen

Eine [IAM-Rolle](#) ist eine Identität in Ihrem AWS-Konto mit spezifischen Berechtigungen. Sie ist einem IAM-Benutzer vergleichbar, ist aber nicht mit einer bestimmten Person verknüpft. Sie können vorübergehend eine IAM-Rolle in der AWS Management Console übernehmen, indem Sie [Rollen wechseln](#). Sie können eine Rolle annehmen, indem Sie eine AWS CLI oder AWS-API-Operation aufrufen oder eine benutzerdefinierte URL verwenden. Weitere Informationen zu Methoden für die Verwendung von Rollen finden Sie unter [Verwenden von IAM-Rollen](#) im IAM-Benutzerhandbuch.

IAM-Rollen mit temporären Anmeldeinformationen sind in folgenden Situationen hilfreich:

- **Verbundbenutzerzugriff:** Um einer Verbundidentität Berechtigungen zuzuweisen, erstellen Sie eine Rolle und definieren Berechtigungen für die Rolle. Wird eine Verbundidentität authentifiziert, so wird die Identität der Rolle zugeordnet und erhält die von der Rolle definierten Berechtigungen. Informationen zu Rollen für den Verbund finden Sie unter [Erstellen von Rollen für externe Identitätsanbieter](#) im IAM-Benutzerhandbuch. Wenn Sie IAM Identity Center verwenden, konfigurieren Sie einen Berechtigungssatz. Wenn Sie steuern möchten, worauf Ihre Identitäten nach der Authentifizierung zugreifen können, korreliert IAM Identity Center den Berechtigungssatz mit einer Rolle in IAM. Informationen zu Berechtigungssätzen finden Sie unter [Berechtigungssätze](#) im AWS IAM Identity Center-Benutzerhandbuch.

- Temporäre IAM-Benutzerberechtigungen: Ein IAM-Benutzer oder eine -Rolle kann eine IAM-Rolle übernehmen, um vorübergehend andere Berechtigungen für eine bestimmte Aufgabe zu erhalten.
- Kontoübergreifender Zugriff – Sie können eine IAM-Rolle verwenden, um einem vertrauenswürdigen Prinzipal in einem anderen Konto den Zugriff auf Ressourcen in Ihrem Konto zu ermöglichen. Rollen stellen die primäre Möglichkeit dar, um kontoübergreifendem Zugriff zu gewähren. In einigen AWS-Services können Sie jedoch eine Richtlinie direkt an eine Ressource anfügen (anstatt eine Rolle als Proxy zu verwenden). Informationen zu den Unterschieden zwischen Rollen und ressourcenbasierten Richtlinien für den kontoübergreifenden Zugriff finden Sie unter [So unterscheiden sich IAM-Rollen von ressourcenbasierten Richtlinien](#) im IAM-Benutzerhandbuch.
- Serviceübergreifender Zugriff: Einige AWS-Services verwenden Features in anderen AWS-Services. Wenn Sie beispielsweise einen Aufruf in einem Service tätigen, führt dieser Service häufig Anwendungen in Amazon EC2 aus oder speichert Objekte in Amazon S3. Ein Dienst kann dies mit den Berechtigungen des aufrufenden Prinzipals mit einer Servicerolle oder mit einer serviceverknüpften Rolle tun.
- Forward access sessions (FAS) – Wenn Sie einen IAM-Benutzer oder eine IAM-Rolle zum Ausführen von Aktionen in AWS verwenden, gelten Sie als Prinzipal. Bei einigen Services könnte es Aktionen geben, die dann eine andere Aktion in einem anderen Service auslösen. FAS verwendet die Berechtigungen des Prinzipals, der einen AWS-Service aufruft, in Kombination mit der Anforderung an den AWS-Service, Anforderungen an nachgelagerte Services zu stellen. FAS-Anfragen werden nur dann gestellt, wenn ein Service eine Anfrage erhält, die eine Interaktion mit anderen AWS-Services oder -Ressourcen erfordert. In diesem Fall müssen Sie über Berechtigungen zum Ausführen beider Aktionen verfügen. Einzelheiten zu den Richtlinien für FAS-Anfragen finden Sie unter [Zugriffssitzungen weiterleiten](#).
- Servicerolle: Eine Servicerolle ist eine [IAM-Rolle](#), die ein Service übernimmt, um Aktionen in Ihrem Namen auszuführen. Ein IAM-Administrator kann eine Servicerolle innerhalb von IAM erstellen, ändern und löschen. Weitere Informationen finden Sie unter [Erstellen einer Rolle zum Delegieren von Berechtigungen an einen AWS-Service](#) im IAM-Benutzerhandbuch.
- Serviceverknüpfte Rolle: Eine serviceverknüpfte Rolle ist ein Typ von Servicerolle, die mit einem AWS-Service verknüpft ist. Der Service kann die Rolle übernehmen, um eine Aktion in Ihrem Namen auszuführen. Serviceverknüpfte Rollen werden in Ihrem AWS-Konto angezeigt und gehören zum Service. Ein IAM-Administrator kann die Berechtigungen für serviceverbundene Rollen anzeigen, aber nicht bearbeiten.
- Anwendungen in Amazon EC2 – Sie können eine IAM-Rolle verwenden, um temporäre Anmeldeinformationen für Anwendungen zu verwalten, die auf einer EC2-Instance ausgeführt

werden und – AWS CLI oder AWS-API-Anforderungen durchführen. Das ist eher zu empfehlen, als Zugriffsschlüssel innerhalb der EC2-Instance zu speichern. Erstellen Sie ein Instance-Profil, das an die Instance angefügt ist, um eine AWS-Rolle einer EC2-Instance zuzuweisen und die Rolle für sämtliche Anwendungen der Instance bereitzustellen. Ein Instance-Profil enthält die Rolle und ermöglicht, dass Programme, die in der EC2-Instance ausgeführt werden, temporäre Anmeldeinformationen erhalten. Weitere Informationen finden Sie unter [Verwenden einer IAM-Rolle zum Erteilen von Berechtigungen für Anwendungen, die auf Amazon EC2-Instances ausgeführt werden](#) im IAM-Benutzerhandbuch.

Informationen dazu, wann Sie IAM-Rollen oder IAM-Benutzer verwenden sollten, finden Sie unter [Erstellen einer IAM-Rolle \(anstatt eines Benutzers\)](#) im IAM-Benutzerhandbuch.

## Funktionsweise von AWS Device Farm mit IAM

Bevor Sie IAM verwenden, um den Zugriff auf Device Farm zu verwalten, sollten Sie verstehen, welche IAM-Features Sie mit Device Farm verwenden können. Einen Überblick über das Zusammenwirken von Device Farm und anderen -AWS-Services mit IAM finden Sie unter [-AWS-Services, die mit IAM funktionieren](#) im IAM-Benutzerhandbuch.

### Themen

- [Identitätsbasierte Richtlinien für Device Farm](#)
- [Ressourcenbasierte Richtlinien für Device Farm](#)
- [Zugriffskontrolllisten](#)
- [Autorisierung basierend auf Device Farm-Tags](#)
- [IAM-Rollen für Device Farm](#)

## Identitätsbasierte Richtlinien für Device Farm

Mit identitätsbasierten IAM-Richtlinien können Sie angeben, welche Aktionen und Ressourcen erteilt oder abgelehnt werden. Darüber hinaus können Sie die Bedingungen festlegen, unter denen Aktionen zugelassen oder abgelehnt werden. Device Farm unterstützt bestimmte Aktionen, Ressourcen und Bedingungsschlüssel. Informationen zu sämtlichen Elementen, die Sie in einer JSON-Richtlinie verwenden, finden Sie in der [IAM-Referenz für JSON-Richtlinienelemente](#) im IAM-Benutzerhandbuch.

## Aktionen

Administratoren können mit AWS-JSON-Richtlinien festlegen, welche Personen zum Zugriff auf welche Ressourcen berechtigt sind. Das heißt, welcher Prinzipal kann Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen.

Das Element `Action` einer JSON-Richtlinie beschreibt die Aktionen, mit denen Sie den Zugriff in einer Richtlinie zulassen oder verweigern können. Richtlinienaktionen haben normalerweise denselben Namen wie die zugehörige AWS-API-Operation. Es gibt einige Ausnahmen, z. B. Aktionen, die nur mit Genehmigung durchgeführt werden können und für die es keine passende API-Operation gibt. Es gibt auch einige Operationen, die mehrere Aktionen in einer Richtlinie erfordern. Diese zusätzlichen Aktionen werden als abhängige Aktionen bezeichnet.

Schließen Sie Aktionen in eine Richtlinie ein, um Berechtigungen zur Durchführung der zugeordneten Operation zu erteilen.

Richtlinienaktionen in Device Farm verwenden das folgende Präfix vor der Aktion: `devicefarm:`. Um beispielsweise jemandem die Berechtigung zum Starten von Selenium-Sitzungen mit dem `Test-CreateTestGridUrlAPI`-Vorgang des Device Farm-Desktop-Browsers zu erteilen, fügen Sie die `devicefarm:CreateTestGridUrl` Aktion in die Richtlinie ein. Richtlinienanweisungen müssen entweder ein `Action`- oder ein `NotAction`-Element enthalten. Device Farm definiert eine eigene Gruppe von Aktionen, die Aufgaben beschreiben, die Sie mit diesem Service durchführen können.

Um mehrere Aktionen in einer einzigen Anweisung anzugeben, trennen Sie sie wie folgt durch Kommata:

```
"Action": [
 "devicefarm:action1",
 "devicefarm:action2"
```

Sie können auch Platzhalter verwenden, um mehrere Aktionen anzugeben. Beispielsweise können Sie alle Aktionen festlegen, die mit dem Wort `List` beginnen, einschließlich der folgenden Aktion:

```
"Action": "devicefarm:List*"
```

Eine Liste der Device Farm-Aktionen finden Sie unter [Von definierte Aktionen AWS Device Farm](#) in der IAM-Service-Autorisierungsreferenz.

## Ressourcen

Administratoren können mit AWS-JSON-Richtlinien festlegen, welche Personen zum Zugriff auf welche Ressourcen berechtigt sind. Das bedeutet die Festlegung, welcher Prinzipal Aktionen für welche Ressourcen unter welchen Bedingungen ausführen kann.

Das JSON-Richtlinienelement `Resource` gibt die Objekte an, auf welche die Aktion angewendet wird. Anweisungen müssen entweder ein `Resource` oder ein `NotResource`-Element enthalten. Als bewährte Methode geben Sie eine Ressource mit dem zugehörigen [Amazon-Ressourcennamen \(ARN\)](#) an. Sie können dies für Aktionen tun, die einen bestimmten Ressourcentyp unterstützen, der als Berechtigungen auf Ressourcenebene bezeichnet wird.

Verwenden Sie für Aktionen, die keine Berechtigungen auf Ressourcenebene unterstützen, z. B. Auflistungsoperationen, einen Platzhalter (\*), um anzugeben, dass die Anweisung für alle Ressourcen gilt.

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

Die Amazon EC2-Instance-Ressource hat den folgenden ARN:

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

Weitere Informationen zum Format von ARNs finden Sie unter [Amazon-Ressourcennamen \(ARNs\) und AWS-Service-Namespaces](#).

Wenn Sie beispielsweise die `i-1234567890abcdef0`-Instance in Ihrer Anweisung angeben möchten, verwenden Sie den folgenden ARN:

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:instance/i-1234567890abcdef0"
```

Um alle Instances anzugeben, die zu einem Konto gehören, verwenden Sie den Platzhalter (\*):

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:instance/*"
```

Einige Device Farm-Aktionen, z. B. das Erstellen von Ressourcen, können nicht für eine Ressource ausgeführt werden. In diesen Fällen müssen Sie den Platzhalter (\*) verwenden.

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

Viele Amazon EC2-API-Aktionen umfassen mehrere Ressourcen. Zum Beispiel wird mit `AttachVolume` einer Instance ein Amazon EBS-Volume angefügt, sodass der IAM-Benutzer über Berechtigungen zur Verwendung des Volumes und der Instance verfügen muss. Um mehrere Ressourcen in einer einzigen Anweisung anzugeben, trennen Sie die ARNs durch Kommata voneinander.

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

Eine Liste der Device Farm-Ressourcentypen und ihrer ARNs finden Sie unter [Von definierte Ressourcentypen AWS Device Farm](#) in der IAM Service Authorization Reference. Informationen zu den Aktionen, mit denen Sie den ARN einzelner Ressourcen angeben können, finden Sie unter [Von definierte Aktionen AWS Device Farm](#) in der IAM-Service-Autorisierungsreferenz.

## Bedingungsschlüssel

Administratoren können mithilfe von AWS-JSON-Richtlinien festlegen, wer zum Zugriff auf was berechtigt ist. Das heißt, welcher Prinzipal kann Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen.

Das Element `Condition` (oder `Condition block`) ermöglicht Ihnen die Angabe der Bedingungen, unter denen eine Anweisung wirksam ist. Das Element `Condition` ist optional. Sie können bedingte Ausdrücke erstellen, die [Bedingungsoperatoren](#) verwenden, z. B. `is equal to` oder `is less than`, damit die Bedingung in der Richtlinie mit Werten in der Anforderung übereinstimmt.

Wenn Sie mehrere `Condition`-Elemente in einer Anweisung oder mehrere Schlüssel in einem einzelnen `Condition`-Element angeben, wertet AWS diese mittels einer logischen AND-Operation aus. Wenn Sie mehrere Werte für einen einzelnen Bedingungsschlüssel angeben, wertet AWS die Bedingung mittels einer logischen OR-Operation aus. Alle Bedingungen müssen erfüllt werden, bevor die Berechtigungen der Anweisung gewährt werden.

Sie können auch Platzhaltervariablen verwenden, wenn Sie Bedingungen angeben. Beispielsweise können Sie einem IAM-Benutzer die Berechtigung für den Zugriff auf eine Ressource nur dann gewähren, wenn sie mit dessen IAM-Benutzernamen gekennzeichnet ist. Weitere Informationen finden Sie unter [IAM-Richtlinienelemente: Variablen und Tags](#) im IAM-Benutzerhandbuch.

AWS unterstützt globale Bedingungsschlüssel und servicespezifische Bedingungsschlüssel. Eine Liste aller globalen AWS-Bedingungsschlüssel finden Sie unter [Globale AWS-Bedingungskontextschlüssel](#) im IAM-Benutzerhandbuch.

Device Farm definiert einen eigenen Satz von Bedingungsschlüsseln und unterstützt auch die Verwendung einiger globaler Bedingungsschlüssel. Eine Liste aller globalen AWS-Bedingungsschlüssel finden Sie unter [Globale AWS-Bedingungskontextschlüssel](#) im IAM-Benutzerhandbuch.

Eine Liste der Device Farm-Bedingungsschlüssel finden Sie unter [Bedingungsschlüssel für AWS Device Farm](#) in der IAM Service Authorization Reference. Informationen dazu, mit welchen Aktionen und Ressourcen Sie einen Bedingungsschlüssel verwenden können, finden Sie unter [Von definierte Aktionen AWS Device Farm](#) in der IAM-Service-Autorisierungs-Referenz.

## Beispiele

Beispiele für identitätsbasierte Device Farm-Richtlinien finden Sie unter [Beispiele für identitätsbasierte AWS Device Farm-Richtlinien](#).

## Ressourcenbasierte Richtlinien für Device Farm

Device Farm unterstützt keine ressourcenbasierten Richtlinien.

## Zugriffskontrolllisten

Device Farm unterstützt keine Zugriffskontrolllisten (ACLs).

## Autorisierung basierend auf Device Farm-Tags

Sie können Tags an Device Farm-Ressourcen anfügen oder Tags in einer Anforderung an Device Farm übergeben. Um den Zugriff auf der Grundlage von Tags zu steuern, geben Sie im Bedingungelement einer [Richtlinie Tag-Informationen](#) an, indem Sie die Schlüssel `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, oder Bedingung `aws:TagKeys` verwenden. Weitere Informationen zum Markieren von Device Farm-Ressourcen finden Sie unter [Taggen in der Gerätefarm](#).

Ein Beispiel für eine identitätsbasierte Richtlinie zur Einschränkung des Zugriffs auf eine Ressource auf der Grundlage der Markierungen dieser Ressource finden Sie unter [Anzeigen von Desktop-Browser-Testprojekten von Device Farm basierend auf Tags](#).

## IAM-Rollen für Device Farm

Eine [IAM-Rolle](#) ist eine Entität in Ihrem AWS Konto mit bestimmten Berechtigungen.

## Verwenden temporärer Anmeldeinformationen mit Device Farm

Device Farm unterstützt die Verwendung temporärer Anmeldeinformationen.

Sie können temporäre Anmeldeinformationen verwenden, um sich mit einem Verbund anzumelden und eine IAM-Rolle oder eine kontoübergreifende Rolle anzunehmen. Sie erhalten temporäre Sicherheitsanmeldeinformationen, indem Sie AWS STS -API-Operationen wie [AssumeRole](#) oder aufrufen [GetFederationToken](#).

### Serviceverknüpfte Rollen

[Serviceverknüpfte Rollen](#) erlauben AWS-Services den Zugriff auf Ressourcen in anderen Services, um eine Aktion in Ihrem Auftrag auszuführen. Serviceverknüpfte Rollen werden in Ihrem IAM-Konto angezeigt und gehören zum Service. Ein IAM-Administrator kann die Berechtigungen für serviceverknüpfte Rollen anzeigen, aber nicht bearbeiten.

Device Farm verwendet serviceverknüpfte Rollen im Desktop-Browser-Testfeature von Device Farm. Weitere Informationen zu diesen Rollen finden Sie unter [Verwenden von serviceverknüpften Rollen im Desktop-Browsertest von Device Farm](#) im Entwicklerhandbuch für .

### Servicerollen

Device Farm unterstützt keine Servicerollen.

Dieses Feature ermöglicht einem Service das Annehmen einer [Servicerolle](#) in Ihrem Namen. Diese Rolle gewährt dem Service Zugriff auf Ressourcen in anderen Diensten, um eine Aktion in Ihrem Namen auszuführen. Servicerollen werden in Ihrem IAM-Konto angezeigt und gehören zum Konto. Dies bedeutet, dass ein IAM-Administrator die Berechtigungen für diese Rolle ändern kann. Dies kann jedoch die Funktionalität des Dienstes beeinträchtigen.

## Verwalten des Zugriffs mit Richtlinien

Für die Zugriffssteuerung in AWS erstellen Sie Richtlinien und weisen diese den AWS-Identitäten oder -Ressourcen zu. Eine Richtlinie ist ein Objekt in AWS, das, wenn es einer Identität oder Ressource zugeordnet wird, deren Berechtigungen definiert. AWS wertet diese Richtlinien aus, wenn ein Prinzipal (Benutzer, Root-Benutzer oder Rollensitzung) eine Anforderung stellt. Berechtigungen in den Richtlinien bestimmen, ob die Anforderung zugelassen oder abgelehnt wird. Die meisten Richtlinien werden in AWS als JSON-Dokumente gespeichert. Weitere Informationen zu Struktur und Inhalten von JSON-Richtliniendokumenten finden Sie unter [Übersicht über JSON-Richtlinien](#) im IAM-Benutzerhandbuch.

Administratoren können mithilfe von AWS-JSON-Richtlinien festlegen, wer zum Zugriff auf was berechtigt ist. Das bedeutet, welcher Prinzipal kann Aktionen für welche Ressourcen und unter welchen Bedingungen ausführen.

Standardmäßig haben Benutzer, Gruppen und Rollen keine Berechtigungen. Ein IAM-Administrator muss IAM-Richtlinien erstellen, die Benutzern die Berechtigung erteilen, Aktionen für die Ressourcen auszuführen, die sie benötigen. Der Administrator kann dann die IAM-Richtlinien zu Rollen hinzufügen, und Benutzer können die Rollen annehmen.

IAM-Richtlinien definieren Berechtigungen für eine Aktion unabhängig von der Methode, die Sie zur Ausführung der Aktion verwenden. Angenommen, es gibt eine Richtlinie, die Berechtigungen für die `iam:GetRole`-Aktion erteilt. Ein Benutzer mit dieser Richtlinie kann Benutzerinformationen über die AWS Management Console, die AWS CLI oder die AWS -API abrufen.

## Identitätsbasierte Richtlinien

Identitätsbasierte Richtlinien sind JSON-Berechtigungsrichtliniendokumente, die Sie einer Identität anfügen können, wie z. B. IAM-Benutzern, -Benutzergruppen oder -Rollen. Diese Richtlinien steuern, welche Aktionen die Benutzer und Rollen für welche Ressourcen und unter welchen Bedingungen ausführen können. Informationen zum Erstellen identitätsbasierter Richtlinien finden Sie unter [Erstellen von IAM-Richtlinien](#) im IAM-Benutzerhandbuch.

Identitätsbasierte Richtlinien können weiter als Inline-Richtlinien oder verwaltete Richtlinien kategorisiert werden. Inline-Richtlinien sind direkt in einen einzelnen Benutzer, eine einzelne Gruppe oder eine einzelne Rolle eingebettet. Verwaltete Richtlinien sind eigenständige Richtlinien, die Sie mehreren Benutzern, Gruppen und Rollen in Ihrem AWS-Konto anfügen können. Verwaltete Richtlinien umfassen von AWS verwaltete und von Kunden verwaltete Richtlinien. Informationen dazu, wie Sie zwischen einer verwalteten Richtlinie und einer eingebundenen Richtlinie wählen, finden Sie unter [Auswahl zwischen verwalteten und eingebundenen Richtlinien](#) im IAM-Benutzerhandbuch.

In der folgenden Tabelle werden die von Device Farm AWS verwalteten Richtlinien beschrieben.

| Änderung                                | Beschreibung                                                       | Datum         |
|-----------------------------------------|--------------------------------------------------------------------|---------------|
| <a href="#">AWSDeviceFarmFullAccess</a> | Bietet vollständigen Zugriff auf alle AWS Device Farm-Operationen. | 15. Juli 2015 |

| Änderung                                            | Beschreibung                                                          | Datum        |
|-----------------------------------------------------|-----------------------------------------------------------------------|--------------|
| <a href="#">AWSServiceRoleForDeviceFarmTestGrid</a> | Ermöglicht Device Farm den Zugriff auf AWS-Ressourcen in Ihrem Namen. | 20. Mai 2021 |

## Weitere Richtlinientypen

AWS unterstützt zusätzliche, weniger häufig verwendete Richtlinientypen. Diese Richtlinientypen können die maximalen Berechtigungen festlegen, die Ihnen von den häufiger verwendeten Richtlinientypen erteilt werden können.

- **Berechtigungsgrenzen:** Eine Berechtigungsgrenze ist ein erweitertes Feature, mit der Sie die maximalen Berechtigungen festlegen können, die eine identitätsbasierte Richtlinie einer IAM-Entität (IAM-Benutzer oder -Rolle) erteilen kann. Sie können eine Berechtigungsgrenze für eine Entität festlegen. Die daraus resultierenden Berechtigungen sind der Schnittpunkt der identitätsbasierten Richtlinien einer Entität und ihrer Berechtigungsgrenzen. Ressourcenbasierte Richtlinien, die den Benutzer oder die Rolle im Feld `Principal` angeben, werden nicht durch Berechtigungsgrenzen eingeschränkt. Eine explizite Zugriffsverweigerung in einer dieser Richtlinien setzt eine Zugriffserlaubnis außer Kraft. Weitere Informationen über Berechtigungsgrenzen finden Sie unter [Berechtigungsgrenzen für IAM-Entitäten](#) im IAM-Benutzerhandbuch.
- **Service-Kontrollrichtlinien (SCPs)** – SCPs sind JSON-Richtlinien, die die maximalen Berechtigungen für eine Organisation oder Organisationseinheit (OE) in AWS Organizations angeben. AWS Organizations ist ein Dienst für die Gruppierung und zentrale Verwaltung mehrerer AWS-Konten Ihres Unternehmens. Wenn Sie innerhalb einer Organisation alle Features aktivieren, können Sie Service-Kontrollrichtlinien (SCPs) auf alle oder einzelne Ihrer Konten anwenden. SCPs schränken Berechtigungen für Entitäten in Mitgliedskonten einschließlich des jeweiligen Root-Benutzer des AWS-Kontos ein. Weitere Informationen zu Organizations und SCPs finden Sie unter [Funktionsweise von SCPs](#) im AWS Organizations-Benutzerhandbuch.
- **Sitzungsrichtlinien:** Sitzungsrichtlinien sind erweiterte Richtlinien, die Sie als Parameter übergeben, wenn Sie eine temporäre Sitzung für eine Rolle oder einen verbundenen Benutzer programmgesteuert erstellen. Die resultierenden Sitzungsberechtigungen sind eine Schnittmenge der auf der Identität des Benutzers oder der Rolle basierenden Richtlinien und der Sitzungsrichtlinien. Berechtigungen können auch aus einer ressourcenbasierten Richtlinie stammen. Eine explizite Zugriffsverweigerung in einer dieser Richtlinien setzt eine Zugriffserlaubnis außer Kraft. Weitere Informationen finden Sie unter [Sitzungsrichtlinien](#) im IAM-Benutzerhandbuch.

## Mehrere Richtlinientypen

Wenn mehrere auf eine Anforderung mehrere Richtlinientypen angewendet werden können, sind die entsprechenden Berechtigungen komplizierter. Informationen dazu, wie AWS die Zulässigkeit einer Anforderung ermittelt, wenn mehrere Richtlinientypen beteiligt sind, finden Sie unter [Logik für die Richtlinienauswertung](#) im IAM-Benutzerhandbuch.

## Beispiele für identitätsbasierte AWS Device Farm-Richtlinien

Standardmäßig haben IAM-Benutzer und -Rollen keine Berechtigung zum Erstellen oder Ändern von Device Farm-Ressourcen. Sie können auch keine Aufgaben ausführen, die die AWS Management Console-, AWS CLI- oder AWS-API benutzen. Ein IAM-Administrator muss IAM-Richtlinien erstellen, die Benutzern und Rollen die Berechtigung zum Ausführen bestimmter API-Operationen für die angegebenen Ressourcen gewähren, die diese benötigen. Der Administrator muss diese Richtlinien anschließend den IAM-Benutzern oder -Gruppen anfügen, die diese Berechtigungen benötigen.

Informationen dazu, wie Sie unter Verwendung dieser beispielhaften JSON-Richtliniendokumente eine identitätsbasierte IAM-Richtlinie erstellen, finden Sie unter [Erstellen von Richtlinien auf der JSON-Registerkarte](#) im IAM-Benutzerhandbuch.

### Themen

- [Bewährte Methoden für Richtlinien](#)
- [Gewähren der Berechtigung zur Anzeige der eigenen Berechtigungen für Benutzer](#)
- [Zugriff auf ein Desktop-Browser-Testprojekt von Device Farm](#)
- [Anzeigen von Desktop-Browser-Testprojekten von Device Farm basierend auf Tags](#)

## Bewährte Methoden für Richtlinien

Identitätsbasierte Richtlinien legen fest, ob jemand Device Farm-Ressourcen in Ihrem Konto erstellen, darauf zugreifen oder sie löschen kann. Dies kann zusätzliche Kosten für Ihr verursachen AWS-Konto. Befolgen Sie beim Erstellen oder Bearbeiten identitätsbasierter Richtlinien die folgenden Anleitungen und Empfehlungen:

- Erste Schritte mit AWS-verwaltete Richtlinien und Umstellung auf Berechtigungen mit den geringsten Berechtigungen: Um Ihren Benutzern und Workloads Berechtigungen zu gewähren, verwenden Sie die AWS-verwaltete Richtlinien die Berechtigungen für viele allgemeine Anwendungsfälle gewähren. Sie sind in Ihrem AWS-Konto verfügbar. Wir empfehlen Ihnen,

die Berechtigungen weiter zu reduzieren, indem Sie vom Kunden verwaltete AWS-Richtlinien definieren, die speziell auf Ihre Anwendungsfälle zugeschnitten sind. Weitere Informationen finden Sie unter [AWS-verwaltete Richtlinien](#) oder [AWS-verwaltete Richtlinien für Auftragsfunktionen](#) im IAM-Benutzerhandbuch.

- Anwendung von Berechtigungen mit den geringsten Rechten: Wenn Sie mit IAM-Richtlinien Berechtigungen festlegen, gewähren Sie nur die Berechtigungen, die für die Durchführung einer Aufgabe erforderlich sind. Sie tun dies, indem Sie die Aktionen definieren, die für bestimmte Ressourcen unter bestimmten Bedingungen durchgeführt werden können, auch bekannt als die geringsten Berechtigungen. Weitere Informationen zur Verwendung von IAM zum Anwenden von Berechtigungen finden Sie unter [Richtlinien und Berechtigungen in IAM](#) im IAM-Benutzerhandbuch.
- Verwenden von Bedingungen in IAM-Richtlinien zur weiteren Einschränkung des Zugriffs: Sie können Ihren Richtlinien eine Bedingung hinzufügen, um den Zugriff auf Aktionen und Ressourcen zu beschränken. Sie können beispielsweise eine Richtlinienbedingung schreiben, um festzulegen, dass alle Anforderungen mithilfe von SSL gesendet werden müssen. Sie können auch Bedingungen verwenden, um Zugriff auf Service-Aktionen zu gewähren, wenn diese durch ein bestimmtes AWS-Service, wie beispielsweise AWS CloudFormation, verwendet werden. Weitere Informationen finden Sie unter [IAM-JSON-Richtlinienelemente: Bedingung](#) im IAM-Benutzerhandbuch.
- Verwenden von IAM Access Analyzer zur Validierung Ihrer IAM-Richtlinien, um sichere und funktionale Berechtigungen zu gewährleisten: IAM Access Analyzer validiert neue und vorhandene Richtlinien, damit die Richtlinien der IAM-Richtliniensprache (JSON) und den bewährten IAM-Methoden entsprechen. IAM Access Analyzer stellt mehr als 100 Richtlinienprüfungen und umsetzbare Empfehlungen zur Verfügung, damit Sie sichere und funktionale Richtlinien erstellen können. Weitere Informationen finden Sie unter [Richtlinienvvalidierung zum IAM Access Analyzer](#) im IAM-Benutzerhandbuch.
- Bedarf einer Multi-Faktor-Authentifizierung (MFA): Wenn Sie ein Szenario haben, das IAM-Benutzer oder Root-Benutzer in Ihrem AWS-Konto erfordert, aktivieren Sie MFA für zusätzliche Sicherheit. Um MFA beim Aufrufen von API-Vorgängen anzufordern, fügen Sie Ihren Richtlinien MFA-Bedingungen hinzu. Weitere Informationen finden Sie unter [Konfigurieren eines MFA-geschützten API-Zugriffs](#) im IAM-Benutzerhandbuch.

Weitere Informationen zu bewährten Methoden in IAM finden Sie unter [Bewährte Methoden für die Sicherheit in IAM](#) im IAM-Benutzerhandbuch.

## Gewähren der Berechtigung zur Anzeige der eigenen Berechtigungen für Benutzer

In diesem Beispiel wird gezeigt, wie Sie eine Richtlinie erstellen, die IAM-Benutzern die Berechtigung zum Anzeigen der eingebundenen Richtlinien und verwalteten Richtlinien gewährt, die ihrer Benutzeridentität angefügt sind. Diese Richtlinie enthält Berechtigungen für die Ausführung dieser Aktion auf der Konsole oder für die programmgesteuerte Ausführung über die AWS CLI oder die AWS-API.

```
{
 "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": "*"
 }
]
}
```

## Zugriff auf ein Desktop-Browser-Testprojekt von Device Farm

In diesem Beispiel möchten Sie einem IAM-Benutzer in Ihrem AWS Konto Zugriff auf eines Ihrer Device Farm-Desktop-Browser-Testprojekte gewähren, `arn:aws:devicefarm:us-west-2:111122223333:testgrid-project:123e4567-e89b-12d3-a456-426655441111`. Sie möchten, dass Elemente im Konto angezeigt werden können, die mit dem Projekt zusammenhängen.

Zusätzlich zum `devicefarm:GetTestGridProject`-Endpunkt muss das Konto über die Endpunkte `devicefarm:ListTestGridSessions`, `devicefarm:GetTestGridSession`, `devicefarm:ListTestGridSessionActions` und `devicefarm:ListTestGridSessionArtifacts` verfügen.

```
{
 "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/*"
 }
]
}
```

Wenn Sie CI-Systeme verwenden, sollten Sie für jeden CI Runner eindeutige Anmeldeinformationen festlegen. Beispielsweise ist es unwahrscheinlich, dass ein CI-System mehr Berechtigungen als

devicefarm:ScheduleRun oder devicefarm:CreateUpload benötigt. Die folgende IAM-Richtlinie beschreibt eine minimale Richtlinie, die es einem CI-Runner ermöglicht, einen Test eines neuen nativen Device Farm-App-Tests zu starten, indem er einen Upload erstellt und ihn zum Planen eines Testlaufs verwendet:

```
{
 "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/*",
]
 }
]
}
```

## Anzeigen von Desktop-Browser-Testprojekten von Device Farm basierend auf Tags

Sie können Bedingungen in Ihrer identitätsbasierten Richtlinie verwenden, um den Zugriff auf Device Farm-Ressourcen basierend auf Tags zu steuern. In diesem Beispiel wird gezeigt, wie Sie eine Richtlinie erstellen, um Projekte und Sitzungen anzuzeigen. Die Berechtigung wird erteilt, wenn das Owner-Tag der angeforderten Ressource mit dem Benutzernamen des anfordernden Kontos übereinstimmt.

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

Sie können diese Richtlinie den IAM-Benutzern in Ihrem Konto anfügen. Wenn ein Benutzer mit dem Namen `richard-roe` versucht, ein Device Farm-Projekt oder eine Sitzung anzuzeigen, muss das Projekt mit dem Tag `Owner=richard-roe` oder `versehen=seinowner=richard-roe` versehen sein. Andernfalls wird dem Benutzer der Zugriff verweigert. Der Tag-Schlüssel `Owner` der Bedingung stimmt sowohl mit `Owner` als auch mit `owner` überein, da die Namen von Bedingungsschlüsseln nicht zwischen Groß- und Kleinschreibung unterscheiden. Weitere Informationen finden Sie unter [IAM-JSON-Richtlinienelemente: Bedingung](#) im IAM-Benutzerhandbuch.

## Fehlerbehebung für AWS Device Farm-Identität und -Zugriff

Verwenden Sie die folgenden Informationen, um häufige Probleme zu diagnostizieren und zu beheben, die bei der Arbeit mit Device Farm und IAM auftreten können.

### Ich bin nicht autorisiert, eine Aktion in Device Farm auszuführen

Wenn Sie die Fehlermeldung erhalten, dass Sie nicht zur Ausführung einer Aktion in der AWS Management Console autorisiert sind, müssen Sie sich an Ihren Administrator wenden. Ihr Administrator ist die Person, die Ihnen Ihren Benutzernamen und Ihr Passwort bereitgestellt hat.

Der folgende Beispielfehler tritt auf, wenn der IAM-Benutzer, `mateojackson`, versucht, die Konsole zu verwenden, um Details zu einer Ausführung anzuzeigen, jedoch nicht über `-devicefarm:GetRunBerechtigungen` verfügt.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
devicefarm:GetRun on resource: arn:aws:devicefarm:us-west-2:123456789101:run:123e4567-
e89b-12d3-a456-426655440000/123e4567-e89b-12d3-a456-426655441111
```

In diesem Fall bittet Mateo den Administrator, die Richtlinien zu aktualisieren, um ihm den Zugriff zu `devicefarm:GetRun` auf der Ressource `arn:aws:devicefarm:us-`

west-2:123456789101:run:123e4567-e89b-12d3-a456-426655440000/123e4567-e89b-12d3-a456-426655441111 über die Aktion `devicefarm:GetRun` zu ermöglichen.

## Ich bin nicht autorisiert, iam durchzuführen:PassRole

Wenn Sie die Fehlermeldung erhalten, dass Sie nicht zur Ausführung der `iam:PassRole` Aktion autorisiert sind, müssen Ihre Richtlinien aktualisiert werden, um eine Rolle an Device Farm übergeben zu können.

Einige AWS-Services erlauben die Übergabe einer vorhandenen Rolle an diesen Dienst, sodass keine neue Servicerolle oder serviceverknüpfte Rolle erstellt werden muss. Hierzu benötigen Sie Berechtigungen für die Übergabe der Rolle an den Dienst.

Der folgende Beispielfehler tritt auf, wenn ein IAM-Benutzer mit dem Namen `marymajor` versucht, die Konsole zu verwenden, um eine Aktion in Device Farm auszuführen. Die Aktion erfordert jedoch, dass der Service über Berechtigungen verfügt, die durch eine Servicerolle gewährt werden. Mary besitzt keine Berechtigungen für die Übergabe der Rolle an den Dienst.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

In diesem Fall müssen die Richtlinien von Mary aktualisiert werden, um die Aktion `iam:PassRole` ausführen zu können.

Wenden Sie sich an Ihren AWS-Administrator, falls Sie weitere Unterstützung benötigen. Ihr Administrator hat Ihnen Ihre Anmeldeinformationen odzur Verfügung gestellt.

## Ich möchte meine Zugriffsschlüssel anzeigen

Nachdem Sie Ihre IAM-Benutzerzugriffsschlüssel erstellt haben, können Sie Ihre Zugriffsschlüssel-ID jederzeit anzeigen. Sie können Ihren geheimen Zugriffsschlüssel jedoch nicht erneut anzeigen. Wenn Sie den geheimen Zugriffsschlüssel verlieren, müssen Sie ein neues Zugriffsschlüsselpaar erstellen.

Zugriffsschlüssel bestehen aus zwei Teilen: einer Zugriffsschlüssel-ID (z. B. `AKIAIOSFODNN7EXAMPLE`) und einem geheimen Zugriffsschlüssel (z. B. `wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY`). Ähnlich wie bei Benutzernamen und Passwörtern müssen Sie die Zugriffsschlüssel-ID und den geheimen Zugriffsschlüssel zusammen verwenden, um Ihre Anforderungen zu authentifizieren. Verwalten Sie Ihre Zugriffsschlüssel so sicher wie Ihren Benutzernamen und Ihr Passwort.

**⚠ Important**

Geben Sie Ihre Zugriffsschlüssel nicht an Dritte weiter, auch nicht für die [Suche nach Ihrer kanonischen Benutzer-ID](#). Wenn Sie dies tun, gewähren Sie anderen Personen möglicherweise den permanenten Zugriff auf Ihr AWS-Konto.

Während der Erstellung eines Zugriffsschlüsselpaars werden Sie aufgefordert, die Zugriffsschlüssel-ID und den geheimen Zugriffsschlüssel an einem sicheren Speicherort zu speichern. Der geheime Zugriffsschlüssel ist nur zu dem Zeitpunkt verfügbar, an dem Sie ihn erstellen. Wenn Sie Ihren geheimen Zugriffsschlüssel verlieren, müssen Sie Ihrem IAM-Benutzer neue Zugriffsschlüssel hinzufügen. Sie können maximal zwei Zugriffsschlüssel besitzen. Wenn Sie bereits zwei Zugriffsschlüssel besitzen, müssen Sie ein Schlüsselpaar löschen, bevor Sie ein neues erstellen. Anweisungen hierfür finden Sie unter [Verwalten von Zugriffsschlüsseln](#) im IAM-Benutzerhandbuch.

Ich bin Administrator und möchte anderen den Zugriff auf Device Farm ermöglichen.

Um anderen Personen oder einer Anwendung Zugriff auf Device Farm zu gewähren, müssen Sie eine IAM-Entität (Benutzer oder Rolle) für die Person oder Anwendung erstellen, die Zugriff benötigt. Sie werden die Anmeldeinformationen für diese Einrichtung verwenden, um auf AWS zuzugreifen. Anschließend müssen Sie der Entität eine Richtlinie anfügen, die dieser die richtigen Berechtigungen in Device Farm gewährt.

Informationen zum Einstieg finden Sie unter [Erstellen Ihrer ersten delegierten IAM-Benutzer und -Gruppen](#) im IAM-Benutzerhandbuch.

Ich möchte Personen außerhalb meines -AWSKontos Zugriff auf meine Device Farm-Ressourcen gewähren

Sie können eine Rolle erstellen, die Benutzer in anderen Konten oder Personen außerhalb Ihrer Organisation für den Zugriff auf Ihre Ressourcen verwenden können. Sie können festlegen, wem die Übernahme der Rolle anvertraut wird. Im Fall von Diensten, die ressourcenbasierte Richtlinien oder Zugriffskontrolllisten (Access Control Lists, ACLs) verwenden, können Sie diese Richtlinien verwenden, um Personen Zugriff auf Ihre Ressourcen zu gewähren.

Weitere Informationen dazu finden Sie hier:

- Informationen dazu, ob Device Farm diese Funktionen unterstützt, finden Sie unter [Funktionsweise von AWS Device Farm mit IAM](#).

- Informationen zum Gewähren des Zugriffs auf Ihre Ressourcen für alle Ihre AWS-Konten finden Sie unter [Gewähren des Zugriffs für einen IAM-Benutzer in einem anderen Ihrer AWS-Konto](#) im IAM-Benutzerhandbuch.
- Informationen dazu, wie Sie AWS-Konten-Drittanbieter Zugriff auf Ihre Ressourcen bereitstellen, finden Sie unter [Gewähren des Zugriffs auf AWS-Konten von externen Benutzern](#) im IAM-Benutzerhandbuch.
- Informationen dazu, wie Sie über einen Identitätsverbund Zugriff gewähren, finden Sie unter [Gewähren von Zugriff für extern authentifizierte Benutzer \(Identitätsverbund\)](#) im IAM-Benutzerhandbuch.
- Informationen zum Unterschied zwischen der Verwendung von Rollen und ressourcenbasierten Richtlinien für den kontoübergreifenden Zugriff finden Sie unter [So unterscheiden sich IAM-Rollen von ressourcenbasierten Richtlinien](#) im IAM-Benutzerhandbuch.

## Compliance-Validierung für AWS Device Farm

Die Auditoren Dritter bewerten die Sicherheit und die Compliance von AWS Device Farm im Rahmen mehrerer AWS-Compliance-Programme. Hierzu zählen unter anderem SOC, PCI, FedRAMP und HIPAA. AWS Device Farm fallen nicht unter AWS-Compliance-Programme.

Eine Liste der AWS-Services im Bereich bestimmter Compliance-Programme finden Sie unter [AWS-Services im Bereich nach Compliance-Programm](#). Allgemeine Informationen finden Sie unter [AWS-Compliance-Programme](#).

Sie können Auditberichte von Drittanbietern unter AWS Artifact herunterladen. Weitere Informationen finden Sie unter [Herunterladen von Berichten in AWS Artifact](#).

Ihre Compliance-Verantwortung bei der Nutzung von Device Farm hängt von der Sensibilität Ihrer Daten, den Compliance-Zielen Ihres Unternehmens und den geltenden Gesetzen und Vorschriften ab. AWS stellt die folgenden Ressourcen zur Verfügung, die Sie bei der Einhaltung der Vorschriften unterstützen:

- [Quick-Start-Anleitungen für Sicherheit und Compliance](#) – In diesen Bereitstellungsleitfäden finden Sie wichtige Überlegungen zur Architektur sowie die einzelnen Schritte zur Bereitstellung von Sicherheits- und Compliance-orientierten Basisumgebungen in AWS.
- [AWS-Compliance-Ressourcen](#) – Diese Arbeitsbücher und Leitfäden könnten für Ihre Branche und Ihren Standort relevant sein.

- [Ressourcen anhand von Regeln bewerten](#) in der AWS Config Leitfaden für Entwickler—AWS Config bewertet, wie gut Ihre Ressourcenkonfigurationen internen Praktiken, Branchenrichtlinien und Vorschriften entsprechen.
- [AWS Security Hub](#) – Dieser AWS-Service liefert einen umfassenden Überblick über den Sicherheitsstatus in AWS. So können Sie die Compliance mit den Sicherheitsstandards in der Branche und den bewährten Methoden abgleichen.

## Datenschutz in AWS Device Farm

Das Modell der AWS geteilten gilt für den Datenschutz in AWS Device Farm (Device Farm ).

<https://aws.amazon.com/compliance/shared-responsibility-model/> Wie in diesem Modell beschrieben, ist AWS für den Schutz der globalen Infrastruktur verantwortlich, in der die gesamte AWS Cloud ausgeführt wird. Sie sind dafür verantwortlich, die Kontrolle über Ihre in dieser Infrastruktur gehosteten Inhalte zu behalten. Sie sind auch für die Sicherheitskonfiguration und die Verwaltungsaufgaben für die von Ihnen verwendeten AWS-Services verantwortlich. Weitere Informationen zum Datenschutz finden Sie unter [Häufig gestellte Fragen zum Datenschutz](#). Informationen zum Datenschutz in Europa finden Sie im Blog-Bertrag [AWS-Modell der geteilten Verantwortung und in der DSGVO](#) im AWS-Sicherheitsblog.

Aus Datenschutzgründen empfehlen wir, AWS-Konto-Anmeldeinformationen zu schützen und einzelne Benutzer mit AWS IAM Identity Center oder AWS Identity and Access Management (IAM) einzurichten. So erhält jeder Benutzer nur die Berechtigungen, die zum Durchführen seiner Aufgaben erforderlich sind. Außerdem empfehlen wir, die Daten mit folgenden Methoden zu schützen:

- Verwenden Sie für jedes Konto die Multi-Faktor Authentifizierung (MFA).
- Verwenden Sie SSL/TLS für die Kommunikation mit AWS-Ressourcen. Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Richten Sie die API und die Protokollierung von Benutzeraktivitäten mit AWS CloudTrail ein.
- Verwenden Sie AWS-Verschlüsselungslösungen zusammen mit allen Standardsicherheitskontrollen in AWS-Services.
- Verwenden Sie erweiterte verwaltete Sicherheitsservices wie Amazon Macie, die dabei helfen, in Amazon S3 gespeicherte persönliche Daten zu erkennen und zu schützen.
- Wenn Sie für den Zugriff auf AWS über eine Befehlszeilenschnittstelle oder über eine API FIPS 140-2-validierte kryptografische Module benötigen, verwenden Sie einen FIPS-Endpunkt. Weitere Informationen über verfügbare FIPS-Endpunkte finden Sie unter [Federal Information Processing Standard \(FIPS\) 140-2](#).

Wir empfehlen dringend, in Freitextfeldern, z. B. im Feld Name, keine vertraulichen oder sensiblen Informationen wie die E-Mail-Adressen Ihrer Kunden einzugeben. Dies gilt auch, wenn Sie mit Device Farm oder anderen AWS-Services über die Konsole, APIAWS CLI, oder AWS SDKs arbeiten. Alle Daten, die Sie in Tags oder Freitextfelder eingeben, die für Namen verwendet werden, können für Abrechnungs- oder Diagnoseprotokolle verwendet werden. Wenn Sie eine URL für einen externen Server bereitstellen, empfehlen wir dringend, keine Anmeldeinformationen zur Validierung Ihrer Anforderung an den betreffenden Server in die URL einzuschließen.

## Verschlüsselung während der Übertragung

Die Device Farm-Endpunkte unterstützen nur signierte HTTPS-Anforderungen (SSL/TLS), sofern nicht anders angegeben. Alle Inhalte, die über Upload-URLs von Amazon S3 abgerufen oder in Amazon S3 platziert werden, werden mit SSL/TLS verschlüsselt. Weitere Informationen zum Anmelden von HTTPS-Anforderungen in AWS finden Sie unter [Signieren von AWS-API-Anforderungen](#) in der allgemeinen AWS-Referenz.

Sie sind verantwortlich für die Verschlüsselung und das Sichern der Kommunikationen von Ihren getesteten Anwendungen und allen Anwendungen, die bei der Ausführung von Tests auf dem Gerät installiert wurden.

## Verschlüsselung im Ruhezustand

Die Desktop-Browser-Testfunktion von Device Farm unterstützt die Verschlüsselung im Ruhezustand für Artefakte, die während Tests generiert wurden.

Die Testdaten für physische Mobilgeräte von Device Farm werden im Ruhezustand nicht verschlüsselt.

## Datenaufbewahrung

Daten in Device Farm werden für einen begrenzten Zeitraum aufbewahrt. Nach Ablauf des Aufbewahrungszeitraums werden die Daten aus dem Device Farm-Unterstützungsspeicher entfernt, aber alle Metadaten (ARNs , Upload-Daten, Dateinamen usw.) werden für die zukünftige Verwendung beibehalten. In der folgenden Tabelle ist der Aufbewahrungszeitraum für verschiedene Inhaltstypen aufgeführt.

| Inhaltstyp               | Aufbewahrungszeitraum (Tage) |
|--------------------------|------------------------------|
| Hochgeladene Anwendungen | 30                           |

| Inhaltstyp                          | Aufbewahrungszeitraum (Tage) |
|-------------------------------------|------------------------------|
| Hochgeladene Testpakete             | 30                           |
| Protokolle                          | 400                          |
| Videoaufnahmen und andere Artefakte | 400                          |

Es liegt in Ihrer Verantwortung, Inhalte zu archivieren, die Sie länger aufbewahren möchten.

## Datenverwaltung

Daten in Device Farm werden je nachdem, welche Funktionen verwendet werden, unterschiedlich verwaltet. In diesem Abschnitt wird erläutert, wie Daten während und nach der Verwendung von Device Farm verwaltet werden.

### Desktop-Browsertests

Instances, die während Selenium-Sitzungen verwendet werden, werden nicht gespeichert. Alle Daten, die durch Browserinteraktionen generiert werden, werden beim Ende der Sitzung verworfen.

Diese Funktion unterstützt derzeit die Verschlüsselung im Ruhezustand für Artefakte, die während des Tests generiert wurden.

### Testen physischer Geräte

In den folgenden Abschnitten finden Sie Informationen über die Schritte, die AWS unternehmen, um Geräte zu bereinigen oder zu beschädigen, nachdem Sie Device Farm verwendet haben.

Die Testdaten für physische Mobilgeräte von Device Farm werden im Ruhezustand nicht verschlüsselt.

### Flotten für öffentliche Geräte

Nachdem die Testausführung abgeschlossen ist, führt Device Farm eine Reihe von Bereinigungsaufgaben auf jedem Gerät in der öffentlichen Geräteflotte durch, einschließlich der Deinstallation Ihrer App. Wenn wir die Deinstallation Ihrer Anwendung oder einen der anderen Bereinigungsschritt nicht überprüfen können, wird das Gerät vor der Wiedereinbetriebnahme zurückgesetzt.

**Note**

Es ist möglich, dass Daten zwischen Sitzungen bestehen bleiben, insbesondere wenn Sie das Gerätesystem außerhalb des Kontexts Ihrer App verwenden. Aus diesem Grund empfehlen wir, während Ihrer automatisierten Test- und Remote-Zugriffssitzungen keine sensiblen Informationen (z. B. Google-Konto oder Apple-ID), persönlichen Informationen und andere sicherheitsrelevante Details einzugeben, da Device Farm Videos und Aktivitätsprotokolle erfasst, die während Ihrer Verwendung jedes Geräts stattfinden.

## Private Geräte

Nach dem Ablauf oder der Beendigung Ihres Vertrags über ein privates Gerät wird das Gerät außer Betrieb genommen und in Übereinstimmung mit AWS-Richtlinien für die Löschung sicher gelöscht. Weitere Informationen finden Sie unter [Arbeiten mit privaten Geräten in AWS Device Farm](#).

## Schlüsselverwaltung

Derzeit bietet Device Farm keine externe Schlüsselverwaltung für die Verschlüsselung von Daten im Ruhezustand oder während der Übertragung an.

## Richtlinie für den Datenverkehr zwischen Netzwerken

Device Farm kann nur für private Geräte so konfiguriert werden, dass Amazon-VPC-Endpunkte verwendet werden, um eine Verbindung zu Ihren Ressourcen in herzustellenAWS. Der Zugriff auf jede nicht öffentliche AWS Infrastruktur, die Ihrem Konto zugeordnet ist (z. B. Amazon EC2Instances ohne öffentliche IP-Adresse), muss einen Amazon-VPC-Endpunkt verwenden. Unabhängig von der VPC-Endpunktkonfiguration isoliert Device Farm Ihren Datenverkehr von anderen Benutzern im gesamten Device Farm-Netzwerk.

Die Sicherheit von Verbindungen außerhalb Ihres AWS-Netzwerks kann nicht gewährleistet werden und es liegt in Ihrer Verantwortung, alle Internetverbindungen zu sichern, die Ihre Anwendungen herstellen.

## Ausfallsicherheit in AWS Device Farm

Im Zentrum der globalen AWS Infrastruktur stehen die AWS-Regionen und Availability Zones (Verfügbarkeitszonen, AZs). AWS -Regionen stellen mehrere physisch getrennte und isolierte

Availability Zones bereit, die über hoch redundante Netzwerke mit niedriger Latenz und hohen Durchsätzen verbunden sind. Mithilfe von Availability Zones können Sie Anwendungen und Datenbanken erstellen und ausführen, die automatisch Failover zwischen Zonen ausführen, ohne dass es zu Unterbrechungen kommt. Availability Zones sind besser verfügbar, fehlertoleranter und skalierbarer als herkömmliche Infrastrukturen mit einem oder mehreren Rechenzentren.

Weitere Informationen über AWS Regionen und Availability Zones finden Sie unter [AWS Globale Infrastruktur](#).

Weil Device Farm verfügbar ist in us-west-2 Nur in der Region wird dringend empfohlen, Sicherungs- und Wiederherstellungsprozesse zu implementieren. Device Farm sollte nicht die einzige Quelle für hochgeladene Inhalte sein.

Device Farm garantiert nicht die Verfügbarkeit öffentlicher Geräte. Diese Geräte werden aufgrund einer Vielzahl von Faktoren wie etwa Ausfallrate und Quarantänestatus in den öffentlichen Gerätepool eingebunden oder daraus entfernt. Wir empfehlen nicht, sich von der Verfügbarkeit eines Geräts im öffentlichen Gerätepool abhängig zu machen.

## Sicherheit der Infrastruktur in AWS Device Farm

Als verwalteter Service ist AWS Device Farm durch die globalen Verfahren zur Gewährleistung der Netzwerksicherheit von AWS geschützt. Informationen zu AWS-Sicherheitsdiensten und wie AWS die Infrastruktur schützt, finden Sie unter [AWS Cloud-Sicherheit](#). Informationen zum Entwerfen Ihrer AWS-Umgebung anhand der bewährten Methoden für die Infrastruktursicherheit finden Sie unter [Infrastrukturschutz](#) im Security Pillar AWS Well-Architected Framework.

Du benutzt AWS veröffentlichte API-Aufrufe für den Zugriff auf Device Farm über das Netzwerk. Kunden müssen Folgendes unterstützen:

- Transport Layer Security (TLS). Wir benötigen TLS 1.2 und empfehlen TLS 1.3.
- Verschlüsselungs-Suiten mit Perfect Forward Secrecy (PFS) wie DHE (Ephemeral Diffie-Hellman) oder ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Die meisten modernen Systemen wie Java 7 und höher unterstützen diese Modi.

Außerdem müssen Anforderungen mit einer Zugriffsschlüssel-ID und einem geheimen Zugriffsschlüssel signiert sein, der einem IAM-Prinzipal zugeordnet ist. Alternativ können Sie mit [AWS Security Token Service](#) (AWS STS) temporäre Sicherheitsanmeldeinformationen erstellen, um die Anforderungen zu signieren.

## Sicherheit der Infrastruktur für Tests physischer Geräte

Geräte werden während des Tests physisch getrennt. Die Netzwerkisolierung verhindert die geräteübergreifende Kommunikation über drahtlose Netzwerke.

Öffentliche Geräte werden gemeinsam genutzt, und Device Farm bemüht sich nach besten Kräften, die Geräte langfristig zu schützen. Bestimmte Aktionen, z. B. Versuche, vollständige Administratorrechte auf einem Gerät zu erwerben (eine Praxis, die als Rooting oder Jailbreak bezeichnet wird), führen dazu, dass öffentliche Geräte isoliert werden. Sie werden automatisch aus dem öffentlichen Pool entfernt und manuell überprüft.

Auf private Geräte kann nur zugegriffen werden von AWS-Konten, die ausdrücklich dazu autorisiert sind. Device Farm isoliert diese Geräte physisch von anderen Geräten und hält sie in einem separaten Netzwerk.

Auf privat verwalteten Geräten können Tests so konfiguriert werden, dass sie einen Amazon VPC-Endpunkt verwenden, um eingehende und ausgehende Verbindungen zu sichern AWS-Konto.

## Infrastruktursicherheit für das Testen von Desktop-Browsern

Wenn Sie die Funktion für Desktop-Browsertests verwenden, werden alle Testsitzungen voneinander getrennt. Selenium-Instances können intern nicht ohne zwischengeschaltete, AWS-externe Dritte kommunizieren.

Der gesamte Datenverkehr geht zu SeleniumWebDriverController müssen über den HTTPS-Endpunkt hergestellt werden, der mit generiert wurde `createTestGridUrl`.

Die Testfunktion für Desktop-Browser unterstützt derzeit keine Amazon VPC-Endpunktkonfiguration. Sie sind dafür verantwortlich, sicherzustellen, dass jede Device Farm-Testinstanz sicheren Zugriff auf die von ihr getesteten Ressourcen hat.

## Analyse und Verwaltung von Konfigurationsschwachstellen in Device Farm

Device Farm ermöglicht es Ihnen, Software auszuführen, die nicht aktiv vom Anbieter gewartet oder gepatcht wird, z. B. vom Betriebssystemanbieter, Hardwareanbieter oder Telefonanbieter. Device Farm bemüht sich nach besten Kräften, die Software auf dem neuesten Stand zu halten, garantiert jedoch nicht, dass eine bestimmte Version der Software auf einem physischen Gerät auf

dem neuesten Stand ist, da sie so konzipiert ist, dass potenziell anfällige Software verwendet werden kann.

Wenn beispielsweise ein Test auf einem Gerät mit Android 4.4.2 durchgeführt wird, garantiert Device Farm nicht, dass das Gerät gegen die [Sicherheitslücke in Android, bekannt als StageFright](#). Es ist Sache des Herstellers (und manchmal auch des Anbieters) des Geräts, Sicherheitsupdates für Geräte bereitzustellen. Bei einer bösartigen Anwendung, die diese Schwachstelle ausnutzt, ist nicht garantiert, dass sie von unserer automatisierten Quarantäne erfasst wird.

Private Geräte werden gemäß Ihrer Vereinbarung mit AWS gewartet.

Device Farm bemüht sich nach besten Kräften, Kundenanwendungen von folgenden Aktionen abzuhalten: Verwurzelung oder Jailbreak. Device Farm entfernt Geräte, die unter Quarantäne gestellt wurden, aus dem öffentlichen Pool, bis sie manuell überprüft wurden.

Sie sind dafür verantwortlich, alle Bibliotheken oder Versionen von Software, die Sie in Ihren Tests verwenden, wie Python-Wheels und Ruby Gems, auf dem neuesten Stand zu halten. Device Farm empfiehlt, dass Sie Ihre Testbibliotheken aktualisieren.

Diese Ressourcen können dazu beitragen, Ihre Testabhängigkeiten auf dem neuesten Stand zu halten:

- Informationen zur Sicherung von Ruby-Gems finden Sie unter [Sicherheitspraktiken](#) auf der RubyGems-Webseite.
- Informationen über das Sicherheitspaket, das von Pipenv verwendet und von der Python Packaging Authority unterstützt wird, um Ihren Abhängigkeitsgraphen auf bekannte Sicherheitslücken zu scannen, finden Sie auf [Erkennung von Sicherheitslücken](#) auf GitHub.
- Informationen zum Maven-Abhängigkeitsprüfer des Open Web Application Security Project (OWASP) finden Sie unter [OWASP DependencyCheck](#) auf der OWASP-Website.

Denken Sie daran, dass selbst wenn ein automatisiertes System keine bekannten Sicherheitsprobleme meldet, dies nicht bedeutet, dass keine vorhanden sein können. Lassen Sie immer Sorgfalt walten, wenn Sie Bibliotheken oder Tools von Drittanbietern verwenden, und überprüfen Sie kryptografische Signaturen, sofern möglich.

## Reaktion auf Vorfälle in Device Farm

Device Farm überwacht Geräte kontinuierlich auf Verhaltensweisen, die auf Sicherheitsprobleme hinweisen könnten. Wenn AWS auf einen Fall aufmerksam wird, in dem auf einem öffentlichen Gerät

gespeicherte Kundendaten wie Testergebnisse oder Dateien für einen anderen Kunden zugänglich sind, kontaktiert AWS die betroffenen Kunden umgehend. Dies geschieht gemäß der in allen AWS-Services geltenden Standardrichtlinien zur Vorfalldreaktion und Berichterstattung.

## Protokollierung und Überwachung in Device Farm

Dieser Dienst unterstützt AWS CloudTrail, das ist ein Dienst, der aufzeichnet AWS-Rufe nach einem AWS-Konto und liefert Protokolldateien an einen Amazon S3-Bucket. Mithilfe von Informationen, die gesammelt wurden von CloudTrail, können Sie feststellen, an welche Anfragen erfolgreich gestellt wurden AWS-Services, wer die Anfrage gestellt hat, wann sie gestellt wurde und so weiter. Um mehr zu erfahren über CloudTrail, einschließlich Informationen darüber, wie Sie es einschalten und Ihre Protokolldateien finden, finden Sie in [AWS CloudTrail Benutzerleitfaden](#).

Für Informationen zur Verwendung CloudTrail mit Device Farm finden Sie unter [Protokollieren von API-Aufrufen von AWS Device Farm mit AWS CloudTrail](#).

## Bewährte Sicherheitsmethoden für Device Farm

Device Farm bietet eine Reihe von Sicherheitsfunktionen, die Sie bei der Entwicklung und Implementierung Ihrer eigenen Sicherheitsrichtlinien berücksichtigen sollten. Die folgenden bewährten Methoden sind allgemeine Richtlinien und keine vollständige Sicherheitslösung. Da diese bewährten Methoden für Ihre Umgebung möglicherweise nicht angemessen oder ausreichend sind, sollten Sie sie als hilfreiche Überlegungen und nicht als bindend ansehen.

- Gewähren Sie jedem System der kontinuierlichen Integration (Continuous Integration System, CI) unter IAM die geringstmöglichen Berechtigungen. Erwägen Sie, temporäre Anmeldeinformationen für jeden CI-Systemtest zu verwenden, so dass ein CI-System auch bei einer Kompromittierung keine falschen Anforderungen stellen kann. Weitere Informationen zu temporären Anmeldeinformationen finden Sie im [IAM-Benutzerhandbuch](#).
- Verwenden Sie adb-Befehle in einer benutzerdefinierten Testumgebung, um von Ihrer Anwendung erstellte Inhalte zu bereinigen. Weitere Informationen über benutzerdefinierte Testumgebungen finden Sie unter [Arbeiten mit benutzerdefinierten Testumgebungen](#).

# Limits in AWS Device Farm

In der folgenden Liste werden die aktuellen AWS Device Farm-Limits beschrieben:

- Die maximale Größe einer App-Datei, die Sie hochladen können, beträgt 4 GB.
- Es gibt keine Einschränkungen hinsichtlich der Anzahl an Geräten, die in einem Testlauf berücksichtigt werden können. Die maximale Anzahl von Geräten, die Device Farm während eines Testlaufs gleichzeitig testet, beträgt jedoch fünf. (Diese Anzahl kann auf Anfrage erhöht werden.)
- Es gibt keine Einschränkungen in Bezug auf die Anzahl an Testläufen, die Sie planen können.
- Für die Länge von Remote-Zugriffssitzungen gibt es eine Beschränkung von 150 Minuten.
- Für die Länge von automatisierten Testläufen gibt es eine Beschränkung von 150 Minuten.
- Die maximale Anzahl von laufenden Aufträgen, einschließlich ausstehender Aufträge in der Warteschlange in Ihrem Konto, beträgt 250. Dies ist ein weiches Limit.
- Es gibt keine Einschränkungen hinsichtlich der Anzahl an Geräten, die in einer Testausführung berücksichtigt werden können. Die Anzahl der Geräte oder Aufträge, auf denen Sie Tests zu einem bestimmten Zeitpunkt parallel ausführen können, entspricht Ihrer Gleichzeitigkeit auf Kontoebene. Die standardmäßige Gleichzeitigkeit auf Kontoebene für die gemessene Verwendung auf AWS Device Farm ist 5. Sie können je nach Anwendungsfall eine Erhöhung dieser Zahl auf einen bestimmten Schwellenwert beantragen. Die standardmäßige Gleichzeitigkeit auf Kontoebene für die nicht gemessene Verwendung entspricht der Anzahl der Slots, die Sie für diese Plattform abonniert haben.

# Tools und Plugins für AWS Device Farm

Dieser Abschnitt enthält Links und Informationen zur Arbeit mit den Tools und Plug-ins von AWS Device Farm. Sie finden Device Farm-Plugins unter [AWS Labs aufGitHub](#).

Wenn Sie ein Android-Entwickler sind, haben wir auch eine [AWS Device Farm-Beispiel-App für Android aufGitHub](#). Sie können die App und die Beispieltests als Referenz für Ihre eigenen Device Farm-Testskripte verwenden.

## Themen

- [Integration von AWS Device Farm mit dem Jenkins CI-Plugin](#)
- [Gradle-Plug-In für AWS Device Farm](#)

## Integration von AWS Device Farm mit dem Jenkins CI-Plugin

Dieses Plugin bietet AWS Device Farm-Funktionen von Ihrem eigenen Jenkins Continuous Integration (CI) -Server aus. Weitere Informationen finden Sie unter [Jenkins \(Software\)](#).

### Note

Um das Jenkins-Plugin herunterzuladen, gehen Sie zu [GitHub](#) und folgen Sie den Anweisungen in [Schritt 1: Installation des Plugins](#).

Dieser Abschnitt enthält eine Reihe von Verfahren zur Einrichtung und Verwendung des Jenkins CI-Plug-ins mit AWS Device Farm.

## Themen

- [Schritt 1: Installation des Plugins](#)
- [Schritt 2: Erstellen einesAWS Identity and Access ManagementBenutzer für Ihr Jenkins CI Plugin](#)
- [Schritt 3: Anleitung zur initialen Konfigurierung](#)
- [Schritt 4: Das Plugin in einem Jenkins-Job verwenden](#)
- [Abhängigkeiten](#)

Die folgenden Bilder zeigen die Funktionen des Jenkins CI-Plug-ins.



Jenkins > Hello World App >

- [Back to Dashboard](#)
- [Status](#)
- [Changes](#)
- [Workspace](#)
- [Build Now](#)
- [Delete Project](#)
- [Configure](#)
- [AWS Device Farm](#)

## Project Hello World App

[Workspace](#)

[Recent Changes](#)

| Build History |                       | <a href="#">trend</a> |
|---------------|-----------------------|-----------------------|
| #19           | Jul 15, 2015 4:25 AM  |                       |
| #18           | Jul 15, 2015 1:35 AM  |                       |
| #17           | Jul 15, 2015 1:21 AM  |                       |
| #16           | Jul 15, 2015 1:06 AM  |                       |
| #15           | Jul 14, 2015 10:55 PM |                       |

[RSS for all](#) [RSS for failures](#)



### Recent AWS Device Farm Results

| Status    | Build Number        | Pass/Warn/Skip/Fail/Error/Stop | Web Report                  |
|-----------|---------------------|--------------------------------|-----------------------------|
| Completed | <a href="#">#19</a> | 12  0  1  1  1  0              | <a href="#">Full Report</a> |
| Completed | <a href="#">#18</a> | 9  0  1  1  1  0               | <a href="#">Full Report</a> |
| Completed | <a href="#">#17</a> | 12  0  1  1  1  0              | <a href="#">Full Report</a> |
| Completed | <a href="#">#16</a> | 12  0  1  1  1  0              | <a href="#">Full Report</a> |
| Completed | <a href="#">#15</a> | 11  0  1  2  1  0              | <a href="#">Full Report</a> |

### Permalinks

- [Last build \(#19\), 41 min ago](#)
- [Last failed build \(#19\), 41 min ago](#)
- [Last unsuccessful build \(#19\), 41 min ago](#)

## Post-build Actions

### Run Tests on AWS Device Farm

refresh

Project  

[Required] Select your AWS Device Farm project.

Device Pool  

[Required] Select your AWS Device Farm device pool.

Application  

[Required] Pattern to find newly built application.

Store test results locally.

### Choose test to run

- Built-in Fuzz
- Appium Java JUnit
- Appium Java TestNG
- Calabash

Features  

[Required] Pattern to find features.zip.

Tags  

[Optional] Tags to pass into Calabash.

- Instrumentation
- Android UI Automator

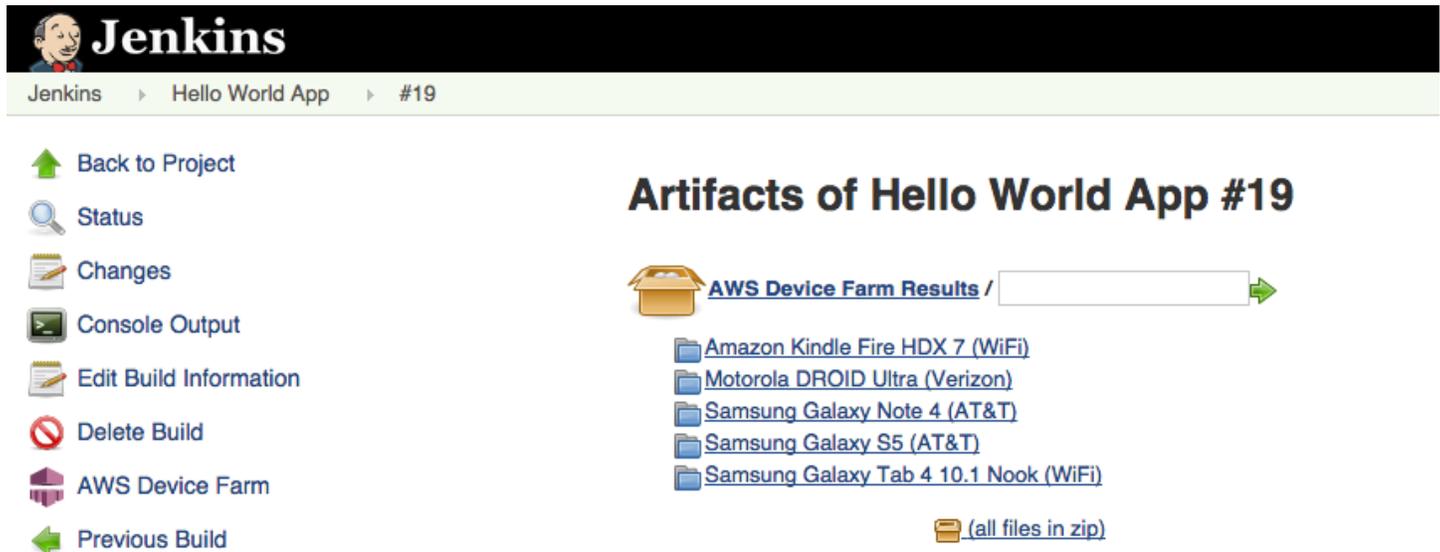
Delete

Add post-build action ▼

Save

Apply

Das Plug-in kann auch alle Testartefakte (Protokolle, Screenshots usw.) lokal abrufen:



Jenkins > Hello World App > #19

Back to Project  
Status  
Changes  
Console Output  
Edit Build Information  
Delete Build  
AWS Device Farm  
Previous Build

## Artifacts of Hello World App #19

 [AWS Device Farm Results](#) /  

- [Amazon Kindle Fire HDX 7 \(WiFi\)](#)
- [Motorola DROID Ultra \(Verizon\)](#)
- [Samsung Galaxy Note 4 \(AT&T\)](#)
- [Samsung Galaxy S5 \(AT&T\)](#)
- [Samsung Galaxy Tab 4 10.1 Nook \(WiFi\)](#)

 [\(all files in zip\)](#)

## Schritt 1: Installation des Plugins

Es gibt zwei Optionen für die Installation des Jenkins Continuous Integration (CI) -Plug-ins für AWS Device Farm. Sie können im Dialogfeld Available Plugins (Verfügbare Plug-ins) in der Web-Benutzeroberfläche von Jenkins nach dem Plug-in suchen oder die Datei `hpi` herunterladen und direkt aus Jenkins heraus installieren.

### Installieren über die Jenkins-Benutzeroberfläche

1. Suchen Sie das Plug-ins über die Jenkins-Benutzeroberfläche, indem Sie nacheinander die Optionen Manage Jenkins (Jenkins verwalten), Manage Plugins (Plug-ins verwalten) und Available (Verfügbar) wählen.
2. Suchen Sie nach `aws-device-farm`.
3. Installieren Sie das AWS Device Farm-Plug-In.
4. Stellen Sie sicher, dass der Besitzer der Datei der Benutzer Jenkins ist.
5. Starten Sie Jenkins neu.

### Laden Sie das Plugin herunter

1. Laden Sie das herunterhpiDatei direkt von <http://updates.jenkins-ci.org/latest/aws-device-farm.hpi>.
2. Stellen Sie sicher, dass der Besitzer der Datei der Benutzer Jenkins ist.

3. Installieren Sie das Plug-in mithilfe einer der folgenden Verfahrenen:
  - Laden Sie das Plug-in hoch, indem Sie nacheinander die Optionen Manage Jenkins (Jenkins verwalten), Manage Plugins (Plug-ins verwalten), Advanced (Erweitert) und dann Upload plugin (Plug-in hochladen) wählen.
  - Verschieben Sie die Datei `hpi` in das Jenkins-Plug-in-Verzeichnis (in der Regel `/var/lib/jenkins/plugins`).
4. Starten Sie Jenkins neu.

## Schritt 2: Erstellen eines AWS Identity and Access Management Benutzer für Ihr Jenkins CI Plugin

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

Informationen zum Erstellen eines neuen IAM-Benutzers finden Sie unter [Einen IAM-Benutzer erstellen \(AWS Management Console\)](#). Achten Sie darauf, dass Sie für jeden Benutzer einen Zugriffsschlüssel erstellen, und laden Sie die Sicherheitsanmeldeinformationen für jeden Benutzer herunter oder speichern Sie diese. Sie benötigen die Anmeldeinformationen zu einem späteren Zeitpunkt.

### Erteilen Sie dem IAM-Benutzer die Erlaubnis, auf Device Farm zuzugreifen

Um dem IAM-Benutzer Zugriff auf Device Farm zu gewähren, erstellen Sie eine neue Zugriffsrichtlinie in IAM und weisen Sie die Zugriffsrichtlinie dann dem IAM-Benutzer wie folgt zu.

#### Note

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

Um die Zugriffsrichtlinie in IAM zu erstellen

1. Öffnen Sie die IAM-Konsole unter <https://console.aws.amazon.com/iam/>.

2. Wählen Sie Policies (Richtlinien).
3. Wählen Sie Create Policy (Richtlinie erstellen) aus. (Wenn die Schaltfläche Get Started (Erste Schritte) angezeigt wird, klicken Sie darauf und wählen Sie anschließend Create Policy (Richtlinie erstellen) aus.)
4. Klicken Sie neben Create Your Own Policy auf Select.
5. Geben Sie unter Policy Name (Richtliniennamen) einen Namen für die Richtlinie ein (z. B. **AWSDeviceFarmAccessPolicy**).
6. Für Beschreibung, geben Sie eine Beschreibung ein, anhand derer Sie diesen IAM-Benutzer Ihrem Jenkins-Projekt zuordnen können.
7. Geben Sie unter Policy Document (Richtliniendokument) die folgende Anweisung ein:

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

8. Wählen Sie Create Policy (Richtlinie erstellen) aus.

Um die Zugriffsrichtlinie dem IAM-Benutzer zuzuweisen

1. Öffnen Sie die IAM-Konsole unter <https://console.aws.amazon.com/iam/>.
2. Wählen Sie Users (Benutzer) aus.
3. Wählen Sie den IAM-Benutzer aus, dem Sie die Zugriffsrichtlinie zuweisen möchten.
4. Wählen Sie im Bereich Permissions (Berechtigungen) unter Managed Policies (Verwaltete Richtlinien) die Option Attach Policy (Richtlinie anfügen).
5. Wählen Sie die gerade erstellte Richtlinie aus (z. B. AWSDeviceFarmAccessPolicy).
6. Wählen Sie Attach Policy.

## Schritt 3: Anleitung zur initialen Konfigurierung

Wenn Sie Ihrer Jenkins-Server zum ersten Mal ausführen möchten, müssen Sie das System wie folgt konfigurieren.

### Note

Wenn Sie [Geräteplätze](#) verwenden, beachten Sie, dass das Geräteplatz-Feature standardmäßig deaktiviert ist.

1. Melden Sie sich an Ihrer Jenkins-Webbenutzeroberfläche an.
2. Wählen Sie auf der linken Seite des Bildschirms die Option Manage Jenkins (Jenkins verwalten) aus.
3. Wählen Sie Configure System (System konfigurieren) aus.
4. Scrollen Sie nach unten zum AWS-Gerätefarm-Kopfzeile.
5. Kopieren Sie Ihre Anmeldeinformationen unter [Schritt 2: Erstellen eines IAM-Benutzers](#) und fügen Sie Ihre Zugriffsschlüssel-ID und den geheimen Zugriffsschlüssel in die jeweiligen Felder ein.
6. Wählen Sie Speichern aus.

## Schritt 4: Das Plugin in einem Jenkins-Job verwenden

Verfahren Sie nach der Installation der Jenkins-Plug-ins wie folgt, um es in einem Jenkins-Auftrag zu verwenden.

1. Melden Sie sich bei Ihrer Jenkins-Webbenutzeroberfläche an.
2. Klicken Sie auf den Auftrag, den Sie bearbeiten möchten.
3. Wählen Sie auf der linken Seite des Bildschirms die Option Configure (Konfigurieren) aus.
4. Führen Sie einen Bildlauf nach unten zur Überschrift Post-build Actions (Postbuild-Aktionen) durch.
5. Klicken Sie Aktion nach dem Build hinzufügen und wählen Sie Führen Sie Tests auf der AWS Device Farm aus.
6. Willen Sie das Projekt aus, das verwendet werden soll.
7. Wählen Sie den Gerätepool aus, der verwendet werden soll.

8. Legen Sie fest, ob die Testartefakte (z. B. Protokolle und Screenshots) lokal archiviert werden sollen.
9. Geben Sie im Feld Application (Anwendung) den Pfad zu Ihrer kompilierten Anwendung an.
10. Wählen Sie den Test aus, den Sie ausführen möchten, und füllen Sie alle Pflichtfelder aus.
11. Wählen Sie Speichern aus.

## Abhängigkeiten

Das Jenkins CI-Plug-in benötigt das AWS Mobile-SDK 1.10.5 oder höher. Weitere Informationen und eine Anleitung zur Installation des SDK finden Sie unter [AWS Mobile-SDK](#).

## Gradle-Plug-In für AWS Device Farm

Dieses Plugin ermöglicht die Integration von AWS Device Farm mit dem Gradle-Build-System in Android Studio. Weitere Informationen finden Sie unter [Gradle](#).

### Note

Um das Gradle-Plugin herunterzuladen, gehen Sie zu [GitHub](#) und folge den Anweisungen in [Erstellen des Device Farm Gradle-Plug-ins](#).

Das Device Farm Gradle Plugin bietet Device Farm-Funktionen aus Ihrer Android Studio-Umgebung. Sie können Tests auf echten Android-Telefonen und -Tablets starten, die von Device Farm gehostet werden.

Dieser Abschnitt enthält eine Reihe von Verfahren zum Einrichten und Verwenden des Device Farm Gradle Plug-ins.

### Themen

- [Schritt 1: Erstellen des AWS Device Farm Gradle-Plug-ins](#)
- [Schritt 2: Einrichten des AWS Device Farm Gradle-Plug-ins](#)
- [Schritt 3: Generieren eines IAM-Benutzers](#)
- [Schritt 4: Konfigurieren von Testtypen](#)
- [Abhängigkeiten](#)

## Schritt 1: Erstellen des AWS Device Farm Gradle-Plug-ins

Dieses Plugin ermöglicht die Integration von AWS Device Farm mit dem Gradle-Build-System in Android Studio. Weitere Informationen finden Sie unter [Gradle](#).

### Note

Das Erstellen des Plug-Ins ist optional. Das Plug-In wird über Maven Central veröffentlicht. Wenn Sie Gradle das direkte Herunterladen des Plug-Ins zulassen möchten, überspringen Sie diesen Schritt und springen Sie zu [Schritt 2: Einrichten des AWS Device Farm Gradle-Plug-ins](#).

So führen Sie das Plug-In aus

1. Gehe zu [GitHub](#) und klonen das Repository.
2. Erstellen Sie das Plug-In mithilfe von `gradle install`.

Das Plug-in wird auf Ihrem lokalen Maven Repository installiert.

Nächster Schritt: [Schritt 2: Einrichten des AWS Device Farm Gradle-Plug-ins](#)

## Schritt 2: Einrichten des AWS Device Farm Gradle-Plug-ins

Wenn noch nicht erfolgt, klonen Sie das Repository und installieren Sie das Plug-In mit dem hier beschriebenen Verfahren: [Erstellen des Device Farm Gradle-Plug-ins](#).

So konfigurieren Sie das AWS Device Farm Gradle-Plugin

1. Fügen Sie das Plug-In-Artefakt Ihrer Abhängigkeitsliste in `build.gradle` hinzu.

```
buildscript {

 repositories {
 mavenLocal()
 mavenCentral()
 }

 dependencies {
 classpath 'com.android.tools.build:gradle:1.3.0'
 }
}
```

```
 classpath 'com.amazonaws:aws-devicefarm-gradle-plugin:1.0'
 }
}
```

2. Konfigurieren Sie das Plug-In in Ihrer `build.gradle`-Datei. Die folgende testspezifische Konfiguration dient als Anleitung:

```
apply plugin: 'devicefarm'

devicefarm {

 // Required. The project must already exist. You can create a project in the
 // AWS Device Farm console.
 projectName "My Project" // required: Must already exist.

 // Optional. Defaults to "Top Devices"
 // devicePool "My Device Pool Name"

 // Optional. Default is 150 minutes
 // executionTimeoutMinutes 150

 // Optional. Set to "off" if you want to disable device video recording during
 // a run. Default is "on"
 // videoRecording "on"

 // Optional. Set to "off" if you want to disable device performance monitoring
 // during a run. Default is "on"
 // performanceMonitoring "on"

 // Optional. Add this if you have a subscription and want to use your unmetered
 // slots
 // useUnmeteredDevices()

 // Required. You must specify either accessKey and secretKey OR roleArn.
 // roleArn takes precedence.
 authentication {
 accessKey "AKIAIOSFODNN7EXAMPLE"
 secretKey "wJalrXUtnFEMI/K7MDENG/bPxrFiCYEXAMPLEKEY"

 // OR

 roleArn "arn:aws:iam::111122223333:role/DeviceFarmRole"
 }
}
```

```
// Optionally, you can
// - enable or disable Wi-Fi, Bluetooth, GPS, NFC radios
// - set the GPS coordinates
// - specify files and applications that must be on the device when your test
runs
devicestate {
 // Extra files to include on the device.
 // extraDataZipFile file("path/to/zip")

 // Other applications that must be installed in addition to yours.
 // auxiliaryApps files(file("path/to/app"), file("path/to/app2"))

 // By default, Wi-Fi, Bluetooth, GPS, and NFC are turned on.
 // wifi "off"
 // bluetooth "off"
 // gps "off"
 // nfc "off"

 // You can specify GPS location. By default, this location is 47.6204,
-122.3491
 // latitude 44.97005
 // longitude -93.28872
}

// By default, the Instrumentation test is used.
// If you want to use a different test type, configure it here.
// You can set only one test type (for example, Calabash, Fuzz, and so on)

// Fuzz
// fuzz { }

// Calabash
// calabash { tests file("path-to-features.zip") }
}
```

3. Führen Sie Ihren Device Farm-Test mit der folgenden Aufgabe aus:`gradle devicefarmUpload`.

In der Build-Ausgabe wird ein Link zur Device Farm-Konsole ausgedruckt, über die Sie Ihre Testausführung überwachen können.

Nächster Schritt: [Generieren eines IAM-Benutzers](#)

## Schritt 3: Generieren eines IAM-Benutzers

AWS Identity and Access Management(IAM) hilft Ihnen bei der Verwaltung von Berechtigungen und Richtlinien für die Arbeit mitAWSRessourcen. In diesem Thema erfahren Sie, wie Sie einen IAM-Benutzer mit Zugriffsberechtigungen für AWS-Gerätefarm-Ressourcen generieren.

Falls Sie dies noch nicht getan haben, führen Sie die Schritte 1 und 2 aus, bevor Sie einen IAM-Benutzer generieren.

Wir empfehlen Ihnen, Ihre nicht zu verwendenAWSRoot-Konto für den Zugriff auf Device Farm. Erstellen Sie stattdessen einen neuen IAM-Benutzer (oder verwenden Sie einen vorhandenen IAM-Benutzer) in IhremAWSKonto und greifen Sie dann mit diesem IAM-Benutzer auf Device Farm zu.

### Note

DasAWSDas Root-Konto oder der IAM-Benutzer, mit dem Sie die folgenden Schritte ausführen, muss berechtigt sein, die folgende IAM-Richtlinie zu erstellen und sie an den IAM-Benutzer anzuhängen. Weitere Informationen finden Sie unter [Arbeiten mit Richtlinien](#).

Um einen neuen Benutzer mit der richtigen Zugriffsrichtlinie in IAM zu erstellen

1. Öffnen Sie die IAM-Konsole unter <https://console.aws.amazon.com/iam/>.
2. Wählen Sie Users (Benutzer) aus.
3. Wählen Sie Create New Users (Neue Benutzer erstellen) aus.
4. Geben Sie den Benutzernamen Ihrer Wahl ein.

Zum Beispiel **GradleUser**.

5. Wählen Sie Erstellen aus.
6. Wählen Sie Download Credentials (Download-Anmeldeinformationen) aus und speichern Sie die Anmeldeinformationen an einem Ort, von dem Sie sie später leicht abrufen können.
7. Wählen Sie Close (Schließen) aus.
8. Wählen Sie den Benutzernamen in der Liste aus.
9. Erweitern Sie unter Permissions (Berechtigungen) die Überschrift Inline Policies (Eingebundene Richtlinien), indem Sie auf den Pfeil nach unten auf der rechten Seite klicken.

10. Wählen Sie hierwo es heißt, Es können keine Inline-Richtlinien angezeigt werden. Um eine zu erstellen, klicken Sie hier.
11. Wählen Sie auf der Seite Set Permissions (Berechtigungen festlegen) die Option Custom Policy (Benutzerdefinierte Richtlinie) aus.
12. Wählen Sie Select (Auswählen).
13. Geben Sie einen Namen für die Richtlinie ein, z. B. **AWSDeviceFarmGradlePolicy**.
14. Fügen Sie die folgende Richtlinie unter Policy Document (Richtliniendokument) ein.

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

15. Klicken Sie auf Apply Policy (Richtlinie anwenden).

Nächster Schritt: [Konfigurieren von Testtypen](#).

Weitere Informationen finden Sie unter [Einen IAM-Benutzer erstellen \(AWS Management Console\)](#) oder [Einrichtung](#).

## Schritt 4: Konfigurieren von Testtypen

Standardmäßig führt das AWS Device Farm Gradle-Plugin den [Arbeiten mit Instrumentierung für Android und AWS Device Farm](#) Test aus. Wenn Sie eigene Tests ausführen oder zusätzliche Parameter angeben möchten, können Sie einen Testtyp konfigurieren. In diesem Thema finden Sie Informationen zu allen verfügbaren Testtypen und wie Sie in Android Studio vorgehen müssen, um sie für die Verwendung zu konfigurieren. Weitere Informationen zu den verfügbaren Testtypen in Device Farm finden Sie unter [Arbeiten mit Testtypen in AWS Device Farm](#).

Wenn noch nicht erfolgt, führen die Schritte 1 bis 3 aus, bevor Sie Testtypen konfigurieren.

**Note**

Wenn Sie [Geräteplätze](#) verwenden, beachten Sie, dass das Geräteplatz-Feature standardmäßig deaktiviert ist.

## Appium

Device Farm bietet Unterstützung für Appium Java JUnit und TestNG für Android.

- [Appium \(unter Java \(JUnit\)\)](#)
- [Appium \(unter Java \(TestNG \)\)](#)

Sie können `useTestNG()` oder `useJUnit()` auswählen. JUnit ist der Standardwert und muss nicht explizit angegeben werden.

```
appium {
 tests file("path to zip file") // required
 useTestNG() // or useJUnit()
}
```

## Integriertes: Fuzz

Device Farm bietet einen integrierten Fuzz-Testtyp, der zufällig Benutzeroberflächenereignisse an Geräte sendet und dann die Ergebnisse meldet.

```
fuzz {

 eventThrottle 50 // optional default
 eventCount 6000 // optional default
 randomizerSeed 1234 // optional default blank

}
```

Weitere Informationen finden Sie unter [Eingebaut: Fuzz \(Android und iOS\)](#).

## Instrumentierung

Device Farm bietet Unterstützung für Instrumentierung (JUnit , BoI, Robotium oder instrumentierungs-basierte Tests) für Android. Weitere Informationen finden Sie unter [Arbeiten mit Instrumentierung für Android und AWS Device Farm](#).

Wenn Sie einen Instrumentierungstest in Gradle ausführen, verwendet Device Farm die aus Ihrem androidTestVerzeichnis generierte .apk Datei als Quelle Ihrer Tests.

```
instrumentation {
 filter "test filter per developer docs" // optional
}
```

## Abhängigkeiten

### Laufzeit

- Das Device Farm Gradle Plugin benötigt dasAWSMobile SDK 1.10.15 oder höher. Weitere Informationen und eine Anleitung zur Installation des SDK finden Sie unter [AWS Mobile-SDK](#).
- Android Tools Builder Test-API 0.5.2
- Apache Commons Lang3 3.3.4

### For Unit Tests (Für Einheitentests)

- Testng 6.8.8
- Jmockit 1.19
- Android Gradle Tools 1.3.0

# Dokumentverlauf

Die folgende Tabelle enthält wichtige Änderungen an der Dokumentation seit der letzten Veröffentlichung dieses Handbuchs.

| Änderung                                                      | Beschreibung                                                                                                                                                                                                                                                                                                                         | Änderungsdatum    |
|---------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|
| AL2-Unterstützung                                             | Device Farm unterstützt jetzt die AL2-Testumgebung für Android. Erfahren Sie mehr über <a href="#">AL2</a> .                                                                                                                                                                                                                         | 6. November 2023  |
| Migration von Standard- zu benutzerdefinierten Testumgebungen | Der <a href="#">Migrationsleitfaden</a> wurde aktualisiert, um die Veralterung von Dokumenten für Tests im Standardmodus im Dezember 2023 zu dokumentieren.                                                                                                                                                                          | 3. September 2023 |
| VPC-ENI-Unterstützung                                         | Device Farm ermöglicht jetzt privaten Geräten die Verwendung der VPC-ENI-Konnektivitätsfunktion, um Kunden dabei zu helfen, eine sichere Verbindung zu ihren privaten Endpunkten herzustellen, die auf AWS, On-Premises-Software oder einem anderen Cloud-Anbieter gehostet werden. Erfahren Sie mehr über <a href="#">VPC-ENI</a> . | 15. Mai 2023      |
| Aktualisierungen der Benutzeroberfläche                       | Die Device Farm-Konsole unterstützt jetzt das Bol-Framework.                                                                                                                                                                                                                                                                         | 28. Juli 2021     |
| Python 3-Unterstützung                                        | Device Farm unterstützt jetzt Python 3 in Tests im benutzerdefinierten Modus. Hier erfahren Sie mehr über die Verwendung von Python 3 in Ihren Testpaketen: <ul style="list-style-type: none"> <li>• <a href="#">Appium (Python)</a></li> <li>• <a href="#">Appium (Python)</a></li> </ul>                                           | 20. April 2020    |
| Neue Sicherheitstinformationen und Informationen zum          | Um die Sicherung von AWS-Services einfacher und umfassender zu gestalten, wurde ein neuer Abschnitt zum Thema Sicherheit erstellt. Weitere Informationen finden Sie unter <a href="#">Sicherheit in AWS Device Farm</a> .                                                                                                            | 27. März 2020     |

| Änderung                                                                                     | Beschreibung                                                                                                                                                                                                                                             | Änderungsdatum    |
|----------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------|
| Taggen von AWS-Ressourcen.                                                                   | Ein neuer Abschnitt zum Markieren in Device Farm wurde hinzugefügt. Weitere Informationen zum Taggen finden Sie unter <a href="#">Taggen in der Gerätefarm</a> .                                                                                         |                   |
| Entfernen von Direct Device Access.                                                          | Direkter Gerätezugriff (Remote-Debugging auf privaten Geräten) ist nicht mehr für die allgemeine Nutzung verfügbar. Wenn Sie Fragen zur künftigen Verfügbarkeit des direkten Gerätezugriffs haben, <a href="#">wenden Sie sich an uns</a> .              | 9. September 2019 |
| Aktualisieren der Gradle-Plug-in-Konfiguration                                               | Eine überarbeitete Gradle-Plugin-Konfiguration enthält jetzt eine anpassbare Version der Gradle-Konfiguration mit optionalen Parametern, die auskommentiert sind. Weitere Informationen zu <a href="#">Einrichtung des Device Farm Gradle-Plug-ins</a> . | 16. August 2019   |
| Neue Anforderung für Testläufe mit XCTest                                                    | Für Testläufe, die das XCTest-Framework verwenden, benötigt Device Farm jetzt ein Anwendungspaket, das zum Testen erstellt wurde. Weitere Informationen zu <a href="#">the section called "XCTest"</a> .                                                 | 4. Februar 2019   |
| Unterstützung für Appium Node.js und Appium Ruby-Testtypen in benutzerdefinierten Umgebungen | Sie können Ihre Tests jetzt in benutzerdefinierten Appium Node.js- und Appium Ruby-Testumgebungen ausführen. Weitere Informationen zu <a href="#">Arbeiten mit Testtypen in AWS Device Farm</a> .                                                        | 10. Januar 2019   |

| Änderung                                                                                                                                                                                                                                                      | Beschreibung                                                                                                                                                                                                                                                                                                                                                                                                                                                                            | Änderungsdatum         |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|
| <p>Unterstützung für Appium-Server Version 1.7.2 in sowohl Standard- als auch benutzerdefinierten Umgebungen. Unterstützung für Version 1.8.1 mithilfe einer benutzerdefinierten Test-Spezifikation-YAML-Datei in einer benutzerdefinierten Testumgebung.</p> | <p>Mit den Appium Server-Versionen 1.7.2, 1.7.1 und 1.6.5 können Sie Ihre Tests jetzt sowohl in Standard- als auch benutzerdefinierten Testumgebungen ausführen. Sie können Ihre Tests auch mit den Versionen 1.8.1 und 1.8.0 unter Verwendung einer benutzerdefinierten Test-Spezifikation-YAML-Datei in einer benutzerdefinierten Testumgebung ausführen. Weitere Informationen zu <a href="#">Arbeiten mit Testtypen in AWS Device Farm</a>.</p>                                     | <p>2. Oktober 2018</p> |
| <p>Benutzerdefinierte Testumgebungen</p>                                                                                                                                                                                                                      | <p>Mit einer benutzerdefinierten Testumgebung können Sie sicherstellen, dass Ihre Tests wie in Ihrer lokalen Umgebung ausgeführt werden. Device Farm bietet jetzt Unterstützung für Live-Protokoll- und Videostreaming, sodass Sie sofortiges Feedback zu Ihren Tests erhalten, die in einer benutzerdefinierten Testumgebung ausgeführt werden. Weitere Informationen zu <a href="#">Arbeiten mit benutzerdefinierten Testumgebungen</a>.</p>                                          | <p>16. August 2018</p> |
| <p>Unterstützung für die Verwendung von Device Farm als AWS CodePipeline Testanbieter</p>                                                                                                                                                                     | <p>Sie können jetzt eine Pipeline in so konfigurieren AWS CodePipeline, dass AWS Device Farm als Testaktionen in Ihrem Release-Prozess verwendet wird. CodePipeline ermöglicht es Ihnen, Ihr Repository schnell zu verknüpfen, um Phasen zu erstellen und zu testen, um ein kontinuierliches Integrationssystem zu erreichen, das auf Ihre Anforderungen zugeschnitten ist. Weitere Informationen zu <a href="#">Verwenden von AWS Device Farm in einem CodePipeline Testphase</a>.</p> | <p>19. Juli 2018</p>   |

| Änderung                                               | Beschreibung                                                                                                                                                                                                                                                                                                                                                                 | Änderungsdatum   |
|--------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------|
| Unterstützung privater Geräte                          | Sie können jetzt private Geräte verwenden, um Testläufe einzuplanen und Sitzungen mit Fernzugriff zu starten. Sie können Profile und Einstellungen für diese Geräte verwalten, Amazon-VPC-Endpunkte zum Testen privater Apps erstellen und Remote-Debugging-Sitzungen erstellen. Weitere Informationen zu <a href="#">Arbeiten mit privaten Geräten in AWS Device Farm</a> . | 2. Mai 2018      |
| Unterstützung für Appium 1.6.3                         | Sie können jetzt die Appium-Version für Ihre benutzerdefinierten Appium-Tests festlegen.                                                                                                                                                                                                                                                                                     | 21. März 2017    |
| Festlegen des Ausführungszeitlimits für einen Testlauf | Sie können die Zeitbeschränkung für die Ausführung eines Testlaufs oder aller Testläufe in einem Projekt festlegen. Weitere Informationen zu <a href="#">Legen Sie das Ausführungszeitlimit für Testläufe in AWS Device Farm fest</a> .                                                                                                                                      | 9. Februar 2017  |
| Traffic Shaping                                        | Sie können jetzt Netzwerkverbindungen und -bedingungen für einen Testlauf simulieren. Weitere Informationen zu <a href="#">Simulieren Sie Netzwerkverbindungen und -bedingungen für Ihre AWS Device Farm Farm-Läufe</a> .                                                                                                                                                    | 8. Dezember 2016 |
| Neuer Abschnitt für Fehlerbehebung                     | Sie können jetzt Probleme mit Testpaket-Uploads beheben, indem Sie eine Reihe von Verfahren zur Behebung von Fehlermeldungen verwenden, die in der Device Farm-Konsole auftreten können. Weitere Informationen zu <a href="#">Fehlerbehebung bei Device Farm-Fehlern</a> .                                                                                                   | 10. August 2016  |
| Remotezugriffssitzungen                                | Sie können remote auf einzelne Geräte in der Konsole zugreifen und mit den Geräten interagieren. Weitere Informationen zu <a href="#">Arbeiten mit Fernzugriff</a> .                                                                                                                                                                                                         | 19. April 2016   |
| Geräteplätze-Self-Service                              | Sie können jetzt Geräteplätze über die AWS Management Console, die AWS Command Line Interface oder die API erwerben. Weitere Informationen finden Sie unter <a href="#">Erwerben Sie einen Geräteslot in Device Farm</a> .                                                                                                                                                   | 22. März 2016    |

| Änderung                                      | Beschreibung                                                                                                                                                                                                         | Änderungsdatum     |
|-----------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|
| So stoppen Sie Testläufe                      | Sie können jetzt Testläufe über die AWS Management Console, die AWS Command Line Interface oder die API anhalten. Weitere Informationen finden Sie unter <a href="#">Stoppen Sie einen Lauf in AWS Device Farm</a> . | 22. März 2016      |
| Neue Testtypen für XCTest-Benutzeroberflächen | Sie können jetzt benutzerdefinierte Tests über XCTest-Benutzeroberflächen in iOS-Anwendungen ausführen. Weitere Informationen über den Testtyp finden Sie unter <a href="#">XCTest-UI</a> .                          | 8. März 2016       |
| Neue Appium Python-Testtypen                  | Sie können jetzt benutzerdefinierte Appium Python-Tests über Android-, iOS- und Webanwendungen ausführen. Weitere Informationen zu <a href="#">Arbeiten mit Testtypen in AWS Device Farm</a> .                       | 19. Januar 2016    |
| Testtypen für Webanwendungen                  | Sie können jetzt benutzerdefinierte Appium Java JUnit- und TestNG-Tests über Webanwendungen ausführen. Weitere Informationen zu <a href="#">Arbeiten mit Web-App-Tests in AWS Device Farm</a> .                      | 19. November 2015  |
| AWS Device Farm Gradle-Plugin                 | Hier erfahren Sie mehr über die Installation und den Betrieb von <a href="#">Gradle-Plugin für Gerätefarm</a> .                                                                                                      | 28. September 2015 |
| Neuer integrierter Test: Explorer             | Bei dem Explorer-Test wird Ihre App durchlaufen, indem jeder einzelne Bildschirm aus der Sicht eines Endbenutzers analysiert und Screenshots erstellt werden.                                                        | 16. September 2015 |
| Unterstützung für iOS hinzugefügt             | Weitere Informationen zum Testen von iOS-Geräten und zum Ausführen von iOS-Tests (einschließlich XCTest) finden Sie unter <a href="#">Arbeiten mit Testtypen in AWS Device Farm</a> .                                | 4. August 2015     |
| Erste veröffentlichte Version                 | Dies ist die erste öffentliche Version des AWS Device Farm-Entwicklerhandbuchs.                                                                                                                                      | 13. Juli 2015      |

# AWS-Glossar

Die neueste AWS-Terminologie finden Sie im [AWS-Glossar](#) in der AWS-Glossar-Referenz.

Die vorliegende Übersetzung wurde maschinell erstellt. Im Falle eines Konflikts oder eines Widerspruchs zwischen dieser übersetzten Fassung und der englischen Fassung (einschließlich infolge von Verzögerungen bei der Übersetzung) ist die englische Fassung maßgeblich.