



Guide du développeur

AWS Device Farm



Version de l'API 2015-06-23

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Device Farm: Guide du développeur

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Les marques commerciales et la présentation commerciale d'Amazon ne peuvent pas être utilisées en relation avec un produit ou un service extérieur à Amazon, d'une manière susceptible d'entraîner une confusion chez les clients, ou d'une manière qui dénigre ou discrédite Amazon. Toutes les autres marques commerciales qui ne sont pas la propriété d'Amazon appartiennent à leurs propriétaires respectifs, qui peuvent ou non être affiliés ou connectés à Amazon, ou sponsorisés par Amazon.

Table of Contents

Qu'est-ce qu'AWS Device Farm ?	1
Accès à distance	1
Tests automatisés des applications	2
Terminologie	2
Configuration	3
Configuration	4
Étape 1 : Inscrivez-vous à AWS	4
Étape 2 : créer ou utiliser un utilisateur IAM dans votre compte AWS	4
Étape 3 : autoriser l'utilisateur IAM à accéder à Device Farm	5
Étape suivante	5
Prise en main	6
Conditions préalables	6
Étape 1 : Se connecter à la console	7
Étape 2 : Création d'un projet	7
Étape 3 : créer et démarrer une course	7
Étape 4 : Afficher les résultats de la course	10
Étapes suivantes	10
Acheter des emplacements pour appareils	11
Acheter des emplacements pour appareils (console)	11
Achetez un emplacement pour appareil (AWS CLI)	13
Acheter un emplacement pour appareil (API)	17
Annulation d'un emplacement pour appareil	17
Annuler un emplacement pour un appareil (console)	18
Annuler un emplacement de périphérique (AWS CLI)	18
Annuler un emplacement de terminal (API)	18
Concepts	19
Devices	19
Appareils pris en charge	20
Pools d'appareils	20
Appareils privés	20
Identification de l'appareil	20
Fentes pour appareils	20
Applications préinstallées pour appareils	21
Fonctionnalités de l'appareil	21

Environnements de test	21
Environnement de test standard	22
Environnement de test personnalisé	22
Exécutions	23
Exécuter la configuration	23
Exécuter la conservation des fichiers	23
Exécuter l'état de l'appareil	24
Sorties en parallèle	24
Configuration du délai d'exécution	24
Publicités en séries	24
Médias en course	24
Tâches courantes pour les courses	24
Applications	24
Applications d'instrumentation	25
Resignature d'applications en plusieurs fois	25
Applications obfusquées en cours d'exécution	25
Rapports	25
Conservation des rapports	26
Composants du rapport	26
Se connecte aux rapports	26
Tâches courantes pour les rapports	26
Séances	26
Appareils compatibles pour l'accès à distance	27
Conservation des fichiers de session	27
Applications d'instrumentation	27
Resignature d'applications dans les sessions	27
Applications masquées dans les sessions	27
Projets	28
Création d'un projet	28
Conditions préalables	28
Création d'un projet (console)	28
Création d'un projet (AWS CLI)	29
Création d'un projet (API)	30
Afficher la liste des projets	30
Prérequis	30
Afficher la liste des projets (console)	30

Afficher la liste des projets (AWS CLI)	31
Afficher la liste des projets (API)	31
Exécutions d'essais	32
Création d'un essai	32
Conditions préalables	33
Création d'un test (console)	33
Créer un test (AWS CLI)	36
Création d'un test (API)	47
Étapes suivantes	47
Configuration du délai d'exécution	48
Conditions préalables	48
Définir le délai d'exécution d'un projet	49
Définissez le délai d'exécution d'un test	49
Simulation des connexions et des conditions du réseau	50
Configurer le modelage du réseau lors de la planification d'un test	50
Création d'un profil réseau	51
Modifier les conditions du réseau pendant votre test	53
Arrêter une course	53
Arrêter une exécution (console)	53
Arrêtez une course (AWS CLI)	55
Arrêter une exécution (API)	57
Afficher la liste des courses	57
Afficher la liste des exécutions (console)	57
Afficher la liste des courses (AWS CLI)	57
Afficher la liste des exécutions (API)	58
Création d'un pool d'appareils	58
Conditions préalables	58
Création d'un pool d'appareils (console)	58
Création d'un pool d'appareils (AWS CLI)	60
Création d'un pool d'appareils (API)	61
Analyse des résultats	61
Affichage des rapports de test	62
Téléchargement d'artefacts	70
Marquage dans Device Farm	76
Balisage de ressources	76
Recherche de ressources par tag	77

Suppression de balises de ressources	78
Cadres de test et tests intégrés	79
Frameworks de test	79
Cadres de test d'applications Android	79
Cadres de test d'applications iOS	79
Cadres de test d'applications Web	80
Frameworks dans un environnement de test personnalisé	80
Support des versions d'Appium	80
Types de tests intégrés	80
Tests Appium automatiques	80
Sélection d'une version d'Appium	81
Sélection d'une WebDriverAgent version pour les tests iOS	82
Intégration aux tests Appium	83
Tests sur Android	98
Cadres de test d'applications Android	99
Types de tests intégrés pour Android	99
Instrumentation	99
Tests iOS	102
Cadres de test d'applications iOS	102
Types de tests intégrés pour iOS	103
XCTest	103
XCTest UI	105
Tests d'applications Web	109
Règles relatives aux appareils mesurés et non mesurés	109
Tests intégrés	110
Intégré : Fuzz (Android et iOS)	110
Environnements de test personnalisés	112
Référence des spécifications de test	113
Flux de travail des spécifications de test	113
Syntaxe des spécifications de test	113
Exemples de spécifications de test	116
Tester les environnements hôtes	131
Hôtes de test disponibles pour les environnements de test personnalisés	131
Sélection d'un hôte de test pour les environnements de test personnalisés	132
Logiciels pris en charge	133
Environnement de test Android	137

Environnement de test iOS	138
Accès à d'autres ressources AWS	144
Aperçu	145
Exigences relatives au rôle IAM	145
Configuration d'un rôle d'exécution IAM	148
Bonnes pratiques	148
Résolution des problèmes	149
Variables d'environnement	149
Variables d'environnement personnalisées	149
Variables d'environnement communes	150
Variables d'environnement pour les tests Appium	151
Variables d'environnement pour les XCUITest tests	152
Bonnes pratiques	153
Migration des tests	155
Considérations relatives à la migration	155
Étapes de la migration	156
Cadre Appium	157
Instrumentation Android	157
Migration des tests iOS XCUITest existants	157
Extension du mode personnalisé	157
Configuration du code PIN d'un appareil	158
Accélérer les tests basés sur Appium	159
Utilisation de webhooks et autres APIs	161
Ajouter des fichiers supplémentaires à votre package de test	163
Accès à distance	167
Création d'une session	167
Conditions préalables	168
Création d'une session à distance	168
Étapes suivantes	182
Utilisation d'une session	183
Conditions préalables	183
Utiliser une session dans la console Device Farm	183
Étapes suivantes	184
Trucs et astuces	184
Récupération des résultats de session	185
Conditions préalables	185

Afficher les détails de la session	185
Téléchargement de vidéos ou de journaux de session	185
Test d'Appium	186
Qu'est-ce qu'un terminal Appium ?	186
Commencer à tester Appium	187
Interaction avec l'appareil à l'aide d'Appium	187
Utilisation d'applications pour les tests avec votre session Appium	188
Comment utiliser le point de terminaison Appium	189
Consulter les journaux de votre serveur Appium	198
Fonctionnalités et commandes Appium prises en charge	210
Fonctionnalités prises en charge	210
Commandes prises en charge	210
Appareils privés	213
Création d'un profil d'instance	214
Demander des appareils privés supplémentaires	216
Création d'un test ou d'une session d'accès à distance	218
Sélection d'appareils privés	219
Règles ARN de l'appareil	220
Règles relatives aux étiquettes des instances de l'appareil	221
Règles ARN de l'instance	221
Création d'un pool d'appareils privé	222
Création d'un pool d'appareils privés avec des appareils privés (AWS CLI)	224
Création d'un pool d'appareils privés avec des appareils privés (API)	225
Ignorer la nouvelle signature d'une application	225
Ignorer la nouvelle signature des applications sur les appareils Android	226
Ignorer la nouvelle signature d'une application sur les appareils iOS	227
Créez une session d'accès à distance pour faire confiance à votre application	227
Amazon VPC dans toutes les régions	229
Vue d'ensemble du peering VPC pour différentes régions VPCs	230
Conditions préalables à l'utilisation d'Amazon VPC	231
Établir une connexion de peering entre deux VPCs	232
Mise à jour des tables de routage pour VPC-1 et VPC-2	232
Création de groupes cibles	233
Créer un Network Load Balancer	235
Créer un service de point de terminaison d'un VPC	236
Création d'une configuration de point de terminaison VPC dans l'application	236

Création d'un essai	237
Création de systèmes VPC évolutifs	237
Résilience d'appareils privés dans Device Farm	237
Connectivité VPC	238
AWS contrôle d'accès et IAM	240
Rôles liés à un service	241
Autorisations de rôle liées à un service pour Device Farm	242
Création d'un rôle lié à un service pour Device Farm	245
Modification d'un rôle lié à un service pour Device Farm	245
Supprimer un rôle lié à un service pour Device Farm	246
Régions prises en charge pour les rôles liés au service Device Farm	246
Conditions préalables	247
Connexion à Amazon VPC	248
Restrictions	250
Utilisation des services de point de terminaison VPC - Legacy	250
Avant de commencer	252
Étape 1 : Création d'un Network Load Balancer	253
Étape 2 : créer un service de point de terminaison VPC	255
Étape 3 : créer une configuration de point de terminaison VPC	256
Étape 4 : Création d'un essai	258
Journalisation des appels d'API AWS CloudTrail avec	259
Informations sur AWS Device Farm dans CloudTrail	259
Comprendre les entrées du fichier journal AWS Device Farm	260
Intégration à AWS Device Farm	263
Configurez CodePipeline pour utiliser vos tests Device Farm	264
AWS CLI référence	268
PowerShell Référence Windows	269
Automatiser Device Farm	270
Exemple : utilisation de la AWS CLI ou du SDK pour télécharger une application ou effectuer un test sur Device Farm	270
Exemple : utilisation du AWS SDK pour démarrer l'exécution d'une Device Farm et collecter des artefacts	284
Résolution des problèmes	289
Résolution des problèmes liés aux applications Android	289
ANDROID_APP_UNZIP_FAILED	290
ANDROID_APP_AAPT_DEBUG_BADGING_FAILED	291

ANDROID_APP_PACKAGE_NAME_VALUE_MISSING	292
ANDROID_APP_SDK_VERSION_VALUE_MISSING	293
ANDROID_APP_AAPT_DUMP_XMLTREE_FAILED	293
ANDROID_APP_DEVICE_ADMIN_PERMISSIONS	295
Certaines fenêtres de mon application Android affichent un écran vide ou noir	296
Résolution des problèmes liés à Appium Java JUnit	297
APPIUM_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED	297
APPIUM_JAVA_JUNIT_TEST_PACKAGE_DEPENDENCY_DIR_MISSING	298
APPIUM_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DIR	299
APPIUM_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING	300
APPIUM_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JAR	301
APPIUM_JAVA_JUNIT_TEST_PACKAGE_JUNIT_VERSION_VALUE_UNKNOWN	303
APPIUM_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION	304
Résolution des problèmes liés à Appium Java Web JUnit	305
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED	305
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_DEPENDENCY_DIR_MISSING	306
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DIR ..	307
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING	308
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JAR	309
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_JUNIT_VERSION_VALUE_UNKNOWN ...	311
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION	312
Résolution des problèmes liés à Appium Java TestNG	314
APPIUM_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED	314
APPIUM_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSING	315
APPIUM_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DIR	316
APPIUM_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING	317
APPIUM_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JAR	318
Résolution des problèmes liés à Appium Java TestNG web	320
APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED	320
APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSING	321
APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DIR	322
APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING	323
APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JAR	324
Résolution des problèmes liés à Appium Python	326
APPIUM_PYTHON_TEST_PACKAGE_UNZIP_FAILED	326
APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING	327

APPIUM_PYTHON_TEST_PACKAGE_INVALID_PLATFORM	329
APPIUM_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING	330
APPIUM_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME	331
APPIUM_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISSING	332
APPIUM_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION	333
APPIUM_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS_FAILED	334
APPIUM_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED	336
APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEELS_INSUFFICIENT	337
Résolution des problèmes liés à Appium Python Web	338
APPIUM_WEB_PYTHON_TEST_PACKAGE_UNZIP_FAILED	339
APPIUM_WEB_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING	340
APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PLATFORM	341
APPIUM_WEB_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING	342
APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME	343
APPIUM_WEB_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISSING	344
APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION	345
APPIUM_WEB_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS_FAILED	346
APPIUM_WEB_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED	348
Dépannage des tests d'instrumentation	349
INSTRUMENTATION_TEST_PACKAGE_UNZIP_FAILED	350
INSTRUMENTATION_TEST_PACKAGE_AAPT_DEBUG_BADGING_FAILED	350
INSTRUMENTATION_TEST_PACKAGE_INSTRUMENTATION_RUNNER_VALUE_MISSING	352
INSTRUMENTATION_TEST_PACKAGE_AAPT_DUMP_XMLTREE_FAILED	353
INSTRUMENTATION_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSING	354
Résolution des problèmes liés aux applications iOS	355
IOS_APP_UNZIP_FAILED	355
IOS_APP_PAYLOAD_DIR_MISSING	356
IOS_APP_APP_DIR_MISSING	357
IOS_APP_PLIST_FILE_MISSING	358
IOS_APP_CPU_ARCHITECTURE_VALUE_MISSING	359
IOS_APP_PLATFORM_VALUE_MISSING	360
IOS_APP_WRONG_PLATFORM_DEVICE_VALUE	361
IOS_APP_FORM_FACTOR_VALUE_MISSING	363
IOS_APP_PACKAGE_NAME_VALUE_MISSING	364
IOS_APP_EXECUTABLE_VALUE_MISSING	366
Résolution des problèmes XCTest	367

XCTEST_TEST_PACKAGE_UNZIP_FAILED	367
XCTEST_TEST_PACKAGE_XCTEST_DIR_MISSING	368
XCTEST_TEST_PACKAGE_PLIST_FILE_MISSING	369
XCTEST_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING	370
XCTEST_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING	371
XCTest Interface utilisateur de dépannage	372
XCTEST_UI_TEST_PACKAGE_UNZIP_FAILED	373
XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_MISSING	374
XCTEST_UI_TEST_PACKAGE_APP_DIR_MISSING	374
XCTEST_UI_TEST_PACKAGE_PLUGINS_DIR_MISSING	375
XCTEST_UI_TEST_PACKAGE_XCTEST_DIR_MISSING_IN_PLUGINS_DIR	376
XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING	377
XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING_IN_XCTEST_DIR	378
XCTEST_UI_TEST_PACKAGE_CPU_ARCHITECTURE_VALUE_MISSING	379
XCTEST_UI_TEST_PACKAGE_PLATFORM_VALUE_MISSING	381
XCTEST_UI_TEST_PACKAGE_WRONG_PLATFORM_DEVICE_VALUE	382
XCTEST_UI_TEST_PACKAGE_FORM_FACTOR_VALUE_MISSING	383
XCTEST_UI_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING	385
XCTEST_UI_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING	386
XCTEST_UI_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSING	388
XCTEST_UI_TEST_PACKAGE_TEST_EXECUTABLE_VALUE_MISSING	389
XCTEST_UI_TEST_PACKAGE_MULTIPLE_APP_DIRS	390
XCTEST_UI_TEST_PACKAGE_MULTIPLE_IPA_DIRS	391
XCTEST_UI_TEST_PACKAGE_BOTH_APP_ET_IPA_DIR_PRESENT	392
XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_PRESENT_DANS_ZIP	393
Sécurité	395
Gestion des identités et des accès	396
Public ciblé	396
Authentification par des identités	396
Comment AWS Device Farm fonctionne avec IAM	397
Gestion de l'accès à l'aide de politiques	402
Exemples de politiques basées sur l'identité	404
Résolution des problèmes	407
Validation de conformité	410
Protection des données	411
Chiffrement en transit	412

Chiffrement au repos	412
Conservation des données	413
Gestion des données	413
Gestion des clés	414
Confidentialité du trafic inter-réseau	414
Résilience	415
Sécurité de l'infrastructure	415
Sécurité de l'infrastructure pour les tests de périphériques physiques	416
Sécurité de l'infrastructure pour les tests de navigateurs de bureau	416
Analyse de la configuration et des vulnérabilités	417
Intervention en cas d'incidents	418
Journalisation et surveillance	418
Bonnes pratiques de sécurité	418
Restrictions	420
Service Limits	420
Limites de fichiers	421
Limites d'API	421
Limites des terminaux Appium	422
Limites de variables d'environnement personnalisées	423
Outils et plugins	424
Plug-in Jenkins CI	424
Dépendances	427
Installation du plugin Jenkins CI	427
Création d'un utilisateur IAM pour votre plugin Jenkins CI	428
Configuration du plugin Jenkins CI pour la première fois	430
Utilisation du plugin dans une tâche Jenkins	431
Plug-in Device Farm Gradle	431
Dépendances	432
Création du plugin Device Farm Gradle	432
Configuration du plugin Device Farm Gradle	433
Génération d'un utilisateur IAM dans le plugin Device Farm Gradle	435
Configuration des types de tests	437
Historique de la documentation	439
AWS Glossaire	445
.....	cdxlv

Qu'est-ce qu'AWS Device Farm ?

Device Farm est un service de test d'applications que vous pouvez utiliser pour tester et interagir avec vos applications Android, iOS et Web sur de vrais téléphones et tablettes physiques hébergés par Amazon Web Services (AWS).

Il existe deux manières principales d'utiliser Device Farm :

- Accédez à distance à un appareil depuis votre ordinateur local, soit de manière interactive dans votre navigateur Web, soit en le testant automatiquement à l'aide d'Appium à partir d'un client local.
- Exécutez automatiquement des tests d'applications à l'aide de l'environnement d'exécution de tests géré de Device Farm.

Note

Device Farm est uniquement disponible dans la région us-west-2 (Oregon).

Accès à distance

L'accès à distance vous permet d'interagir avec un appareil via votre navigateur Web en temps réel. L'accès à distance vous permet également d'exécuter des tests Appium depuis votre client local sur des appareils Device Farm distants à l'aide d'un point de terminaison Appium géré.

L'interaction en temps réel avec un appareil peut être utile dans de nombreux scénarios, tels que le test manuel d'applications, la reproduction de bogues sur un appareil spécifique, la vérification du rendu visuel de votre application sur différents types d'écran et les séquences d'installation et de mise à niveau de l'application. Le point de terminaison Appium entièrement géré de Device Farm vous permet de développer, de tester et de déboguer vos tests Appium, en fournissant des commentaires rapides.

[Le point de terminaison Appium prend en charge le langage de votre choix, n'importe quel IDE local, le débogage en direct avec points d'arrêt, la vidéo et les journaux en direct, ainsi que des outils tels qu'Appium Inspector.](#) Vous pouvez exécuter des tests autant de fois que vous le souhaitez sur le même appareil pendant votre session d'accès à distance, dans la [limite de 150 minutes](#).

Au cours d'une session d'accès à distance, Device Farm enregistre les détails des actions qui ont lieu lorsque vous interagissez avec l'appareil. Des journaux contenant ces informations et une capture vidéo de la session sont générés à la fin de la session.

Tests automatisés des applications

Device Farm vous permet d'exécuter des tests automatisés sur plusieurs appareils en parallèle en téléchargeant votre application et vos tests. Les tests sont exécutés automatiquement dans un environnement entièrement géré sur des hôtes de test sur lesquels vous pouvez configurer [un fichier de spécifications de test](#). L'environnement utilise les [hôtes de test](#) de Device Farm, vous n'avez donc pas à vous soucier de configurer votre propre infrastructure pour exécuter les tests. Les hôtes et appareils de test peuvent se connecter en toute sécurité à votre VPC pour accéder à vos points de terminaison privés.

Au fur et à mesure que les tests sont terminés, un rapport de test est généré qui contient des résultats de haut niveau, des journaux de bas niveau, des captures d'écran et vos artefacts de test.

Device Farm prend en charge les tests d'applications Android et iOS natives et hybrides. Pour plus d'informations sur les types de tests pris en charge, consultez [Frameworks de test et tests intégrés dans AWS Device Farm](#).

Terminologie

Device Farm introduit les termes suivants qui définissent la manière dont les informations sont organisées :

groupe d'appareils

Ensemble d'appareils qui partagent généralement des caractéristiques similaires, comme une plateforme, un fabricant ou un modèle.

tâche

Demande adressée à Device Farm pour tester une seule application sur un seul appareil. Une tâche contient une ou plusieurs suites.

mesure

Fait référence à la facturation des appareils. Vous pouvez voir des références à des appareils limités ou illimités dans la documentation et le document de référence d'API. Pour plus d'informations sur les tarifs, consultez la section [Tarifs d'AWS Device Farm](#).

project

Espace de travail logique contenant des exécutions, une par test d'une seule application par rapport à un ou plusieurs appareils. Vous pouvez utiliser des projets pour organiser les espaces de travail comme vous le souhaitez. Par exemple, vous pouvez avoir un projet par titre d'application ou un projet par plateforme. Vous pouvez créer autant de projets que vous en avez besoin.

report

Contient des informations sur une exécution, c'est-à-dire une demande adressée à Device Farm pour tester une seule application sur un ou plusieurs appareils. Pour de plus amples informations, veuillez consulter [Rapports dans AWS Device Farm](#).

run

Build spécifique de votre application, avec un ensemble de tests spécifique, à exécuter sur un ensemble d'appareils spécifique. Une exécution génère un rapport de résultats. Une exécution contient une ou plusieurs tâches. Pour de plus amples informations, veuillez consulter [Exécutions](#).

séance

Interaction en temps réel avec un appareil physique réel via votre navigateur web. Pour de plus amples informations, veuillez consulter [Séances](#).

suite

Organisation hiérarchique des tests dans un package de test. Une suite contient un ou plusieurs tests.

test

Cas de test individuel dans un package de test.

Pour plus d'informations sur Device Farm, consultez [Concepts](#).

Configuration

Pour utiliser Device Farm, voir [Configuration](#).

Configuration d'AWS Device Farm

Avant d'utiliser Device Farm pour la première fois, vous devez effectuer les tâches suivantes :

Rubriques

- [Étape 1 : Inscrivez-vous à AWS](#)
- [Étape 2 : créer ou utiliser un utilisateur IAM dans votre compte AWS](#)
- [Étape 3 : autoriser l'utilisateur IAM à accéder à Device Farm](#)
- [Étape suivante](#)

Étape 1 : Inscrivez-vous à AWS

Inscrivez-vous à Amazon Web Services (AWS).

Si vous n'en avez pas Compte AWS, procédez comme suit pour en créer un.

Pour vous inscrire à un Compte AWS

1. Ouvrez l'<https://portal.aws.amazon.com/billing/inscription>.
2. Suivez les instructions en ligne.

Dans le cadre de la procédure d'inscription, vous recevrez un appel téléphonique ou un SMS et vous saisirez un code de vérification en utilisant le clavier numérique du téléphone.

Lorsque vous vous inscrivez à un Compte AWS, un Utilisateur racine d'un compte AWS est créé. Par défaut, seul l'utilisateur racine a accès à l'ensemble des Services AWS et des ressources de ce compte. La meilleure pratique de sécurité consiste à attribuer un accès administratif à un utilisateur, et à utiliser uniquement l'utilisateur racine pour effectuer les [tâches nécessitant un accès utilisateur racine](#).

Étape 2 : créer ou utiliser un utilisateur IAM dans votre compte AWS

Nous vous recommandons de ne pas utiliser votre compte AWS root pour accéder à Device Farm. Créez plutôt un utilisateur Gestion des identités et des accès AWS (IAM) (ou utilisez-en un existant) dans votre AWS compte, puis accédez à Device Farm avec cet utilisateur IAM.

Pour plus d'informations, consultez [Création d'un utilisateur IAM \(AWS Management Console\)](#).

Étape 3 : autoriser l'utilisateur IAM à accéder à Device Farm

Donnez à l'utilisateur IAM l'autorisation d'accéder à Device Farm. Pour ce faire, créez une politique d'accès dans IAM, puis attribuez-la à l'utilisateur IAM, comme suit.

Note

Le compte AWS root ou l'utilisateur IAM que vous utilisez pour effectuer les étapes suivantes doit être autorisé à créer la politique IAM suivante et à l'associer à l'utilisateur IAM. Pour plus d'informations, consultez [Utilisation de stratégies](#).

1. Créez une stratégie avec le corps JSON suivant. Donnez-lui un titre descriptif, tel que *DeviceFarmAdmin*.

Pour plus d'informations sur la création de politiques IAM, consultez la section [Création de politiques IAM](#) dans le guide de l'utilisateur IAM.

2. Attachez la politique IAM que vous avez créée à votre nouvel utilisateur. Pour plus d'informations sur l'attachement des politiques IAM aux utilisateurs, consultez la section [Ajouter et supprimer des politiques IAM](#) dans le guide de l'utilisateur IAM.

L'attachement de la politique permet à l'utilisateur IAM d'accéder à toutes les actions et ressources de Device Farm associées à cet utilisateur IAM. Pour plus d'informations sur la façon de restreindre les utilisateurs IAM à un ensemble limité d'actions et de ressources Device Farm, consultez [Gestion des identités et des accès dans AWS Device Farm](#).

Étape suivante

Vous êtes maintenant prêt à commencer à utiliser Device Farm. Consultez [Commencer à utiliser Device Farm](#).

Commencer à utiliser Device Farm

Cette procédure pas à pas vous explique comment utiliser Device Farm pour tester une application Android ou iOS native. Vous utilisez la console Device Farm pour créer un projet, télécharger un fichier .apk ou .ipa, exécuter une série de tests standard, puis afficher les résultats.

Note

Device Farm est disponible uniquement dans la AWS région us-west-2 (Oregon).

Rubriques

- [Conditions préalables](#)
- [Étape 1 : Se connecter à la console](#)
- [Étape 2 : Création d'un projet](#)
- [Étape 3 : créer et démarrer une course](#)
- [Étape 4 : Afficher les résultats de la course](#)
- [Étapes suivantes](#)

Conditions préalables

Avant de commencer, assurez-vous de respecter les prérequis suivants :

- Suivez les étapes de [Configuration](#). Vous avez besoin d'un AWS compte et d'un utilisateur Gestion des identités et des accès AWS (IAM) autorisé à accéder à Device Farm.
- Pour Android, vous pouvez apporter un fichier .apk (package d'application Android) ou utiliser l'exemple d'application que nous fournissons. Pour iOS, vous avez besoin d'un fichier .ipa (archive d'application iOS). Vous téléchargerez le fichier sur Device Farm plus tard dans cette procédure pas à pas.

Note

Assurez-vous que votre fichier .ipa est conçu pour un appareil iOS et non pour un simulateur.

- (Facultatif) Vous devez effectuer un test à partir de l'un des frameworks de test pris en charge par Device Farm. Vous téléchargez ce package de test sur Device Farm, puis vous exécutez le test plus tard dans cette procédure pas à pas. Si aucun package de test n'est disponible, vous pouvez spécifier et exécuter une suite de tests intégrée standard. Pour de plus amples informations, veuillez consulter [Frameworks de test et tests intégrés dans AWS Device Farm](#).

Étape 1 : Se connecter à la console

Vous pouvez utiliser la console Device Farm pour créer et gérer des projets et des exécutions à des fins de test. Vous découvrirez les projets et les exécutions ultérieurement dans cette procédure.

- Connectez-vous à la console Device Farm à l'adresse <https://console.aws.amazon.com/devicefarm>.

Étape 2 : Création d'un projet

Pour tester une application dans Device Farm, vous devez d'abord créer un projet.

1. Dans le volet de navigation, choisissez Mobile Device Testing, puis Projects.
2. Sous Projets de test d'appareils mobiles, choisissez Créer un projet.
3. Sous Créer un projet, entrez un nom de projet (par exemple, **MyDemoProject**).
4. Choisissez Créer.

La console ouvre la page Tests automatisés du projet que vous venez de créer.

Étape 3 : créer et démarrer une course

Maintenant que vous disposez d'un projet, vous pouvez créer et démarrer une exécution. Pour de plus amples informations, veuillez consulter [Exécutions](#).

1. Dans l'onglet Tests automatisés, choisissez Create run. Vous pouvez également suivre le didacticiel intégré à la console en sélectionnant Créer une exécution avec le didacticiel.
2. (Facultatif) Sous Paramètres d'exécution, dans la section Nom de l'exécution, entrez le nom de votre course. Si aucun nom n'est fourni, la console Device Farm nommera votre course « My Device Farm run » par défaut.

3. Sous Paramètres d'exécution, dans la section Type d'exécution, sélectionnez votre type d'exécution. Sélectionnez une application Android si aucune application n'est prête à être testée ou si vous testez une application Android (.apk). Sélectionnez une application iOS si vous testez une application iOS (.ipa).
4. Sous Sélectionner une application, dans la section Options de sélection des applications, choisissez Sélectionner un exemple d'application fourni par Device Farm si aucune application n'est disponible pour les tests. Si vous apportez votre propre application, sélectionnez Télécharger votre propre application, puis choisissez votre fichier de candidature. Si vous chargez une application iOS, veillez à choisir iOS device (Appareil iOS) et non un simulateur.
5. Sous Configurer le test, dans la section Sélectionner le cadre de test, choisissez l'un des frameworks de test ou des suites de tests intégrées. Pour plus d'informations sur chaque option, consultez [Cadres de test et tests intégrés](#).
 - Si vous n'avez pas encore empaqueté vos tests pour Device Farm, choisissez Built-in : Fuzz pour exécuter une suite de tests standard intégrée. Vous pouvez conserver les valeurs par défaut pour Event Count, Event Throttle et Randomizer Seed. Pour de plus amples informations, veuillez consulter [the section called "Intégré : Fuzz \(Android et iOS\)"](#).
 - Si vous disposez d'un package de test issu de l'un des frameworks de test pris en charge, choisissez le framework de test correspondant, puis téléchargez le fichier contenant vos tests.
6. Sous Sélectionner les appareils, choisissez Utiliser le pool de périphériques et Top Devices.
7. (Facultatif) Pour ajouter une configuration supplémentaire, ouvrez le menu déroulant Configuration supplémentaire. Dans cette section, vous pouvez effectuer l'une des opérations suivantes :
 - Pour fournir d'autres données à utiliser par Device Farm pendant l'exécution, à côté de Ajouter des données supplémentaires, choisissez Choose File, puis naviguez jusqu'au fichier .zip contenant les données et sélectionnez-le.
 - Pour installer une application supplémentaire que Device Farm pourra utiliser pendant l'exécution, à côté de Installer d'autres applications, choisissez Choose File, puis naviguez jusqu'au fichier .apk ou .ipa qui contient l'application et sélectionnez-le. Répétez cette procédure pour les autres applications que vous voulez installer. Vous pouvez modifier l'ordre d'installation en faisant glisser et en déposant les applications après les avoir chargées.
 - Pour spécifier si le Wi-Fi, le Bluetooth, le GPS ou le NFC doivent être activés pendant l'exécution, en regard de Set radio states (Définir les états radio), cochez les cases appropriées.

- Pour prédéfinir la latitude et la longitude de l'appareil pour l'exécution, saisissez les coordonnées en regard de Device location (Emplacement de l'appareil).
- Pour prédéfinir les paramètres régionaux de l'appareil pour l'exécution, dans Paramètres régionaux de l'appareil, choisissez les paramètres régionaux.
- Sélectionnez Activer l'enregistrement vidéo pour enregistrer une vidéo pendant le test.
- Sélectionnez Activer la capture des données de performance des applications pour capturer les données de performance de l'appareil.

 Note

Le réglage de l'état radio et des paramètres régionaux de l'appareil ne sont actuellement disponibles que pour les tests natifs d'Android.

 Note

Si vous avez des appareils privés, la configuration spécifique aux appareils privés est également affichée.

8. Au bas de la page, choisissez Create run pour planifier l'exécution.

Device Farm démarre l'exécution dès que les appareils sont disponibles, généralement en quelques minutes. Pour consulter le statut de l'exécution, sur la page Tests automatisés de votre projet, choisissez le nom de votre exécution. Sur la page d'exécution, sous Appareils, chaque appareil commence par l'icône en attente



dans le tableau des appareils, puis passe à l'icône en cours



lorsque le test commence. À la fin de chaque test, la console affiche une icône de résultat du test à côté du nom de l'appareil. Lorsque tous les tests sont terminés, l'icône en attente à côté de l'exécution devient une icône de résultat de test.

Étape 4 : Afficher les résultats de la course

Pour afficher les résultats des tests depuis l'exécution, sur la page Tests automatisés de votre projet, choisissez le nom de votre exécution. Une page récapitulative affiche :

- Le nombre total de tests, par résultat.
- Des listes des tests comportant des avertissements ou des échecs uniques.
- Une liste des appareils avec les résultats des tests pour chacun d'entre eux.
- Toutes les captures d'écran effectuées pendant l'exécution, regroupées par appareil.
- Section permettant de télécharger le résultat de l'analyse.

Pour de plus amples informations, veuillez consulter [Afficher les rapports de test dans Device Farm](#).

Étapes suivantes

Pour plus d'informations sur Device Farm, consultez [Concepts](#).

Acheter un emplacement pour appareil dans Device Farm

Vous pouvez utiliser la console Device Farm AWS Command Line Interface (AWS CLI) ou l'API Device Farm pour acheter un emplacement pour appareil.

Acheter des emplacements pour appareils (console)

1. Connectez-vous à la console Device Farm à l'adresse <https://console.aws.amazon.com/devicefarm>.
2. Dans le volet de navigation, choisissez Mobile Device Testing, puis Device slots.
3. Sur la page Acheter et gérer les emplacements d'appareils, vous pouvez créer votre propre package personnalisé en choisissant le nombre d'emplacements pour les tests automatisés et les appareils d'accès à distance que vous souhaitez acheter. Spécifiez le montant des créneaux pour la période de facturation en cours et pour la période suivante.

Lorsque vous modifiez le montant du créneau, le texte est mis à jour de manière dynamique avec le montant de facturation. Pour plus d'informations, consultez la [tarification d'AWS Device Farm](#).

Important

Si vous modifiez le nombre d'emplacements pour appareils mais que vous voyez un message « Contactez-nous » ou « Contactez-nous pour acheter », cela signifie que votre AWS compte n'est pas encore approuvé pour acheter le nombre d'emplacements que vous avez demandé.

Ces options vous invitent à envoyer un e-mail à l'équipe d'assistance de Device Farm. Dans l'e-mail, indiquez le numéro de chaque type d'appareil que vous souhaitez acheter et pour quel cycle de facturation.

Note

Les modifications apportées aux emplacements des appareils s'appliquent à l'ensemble de votre compte et concernent tous les projets.

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

0

Android slots

0

iOS slots

Next billing period

From August 16, you will have

0

Android slots

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

0

Android slots

0

iOS slots

Next billing period

From August 16, you will have

0

Android slots

0

iOS slots

Save

4. Choisissez Purchase (Acheter). Une fenêtre de confirmation d'achat s'affichera. Vérifiez les informations, puis choisissez Confirmer pour terminer la transaction.

Confirm purchase

×

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

Cancel

Confirm

Sur la page Acheter et gérer les emplacements pour appareils, vous pouvez voir le nombre d'emplacements dont vous disposez actuellement. Si vous avez augmenté ou diminué le nombre d'emplacements, le nombre d'emplacements dont vous disposerez un mois après la date de votre modification s'affiche.

Achetez un emplacement pour appareil (AWS CLI)

Pour acheter l'offre, vous pouvez exécuter la commande `purchase-offering`.

Pour répertorier les paramètres de votre compte Device Farm, y compris le nombre maximum d'emplacements que vous pouvez acheter et le nombre de minutes d'essai gratuites restantes, exécutez la `get-account-settings` commande. Des résultats similaires à ce qui suit s'affichent :

```
{
  "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
    }
  }
}
```

Pour obtenir une liste des offres d'emplacements d'appareils dont vous pouvez bénéficier, exécutez la commande `list-offerings`. Vous devez voir des résultats similaires à ce qui suit :

```
{
  "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"
    }
]
}

```

Pour répertorier les offres promotionnelles disponibles, exécutez la `list-offering-promotions` commande.

Note

Cette commande renvoie uniquement les promotions que vous n'avez pas encore achetées. Dès que vous achetez un ou plusieurs emplacements dans n'importe quelle offre en bénéficiant d'une promotion, celle-ci ne s'affiche plus dans les résultats.

Vous devez voir des résultats similaires à ce qui suit :

```

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

```

Pour obtenir le statut des offres, exécutez la commande `get-offering-status`. Vous devez voir des résultats similaires à ce qui suit :

```
{
  "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
    }
  }
}
```

```
        "quantity": 1
      }
    }
  }
```

Les `list-offering-transactions` commandes `renew-offering` et sont également disponibles pour cette fonctionnalité. Pour de plus amples informations, veuillez consulter [AWS CLI référence](#).

Acheter un emplacement pour appareil (API)

1. Appelez l'[GetAccountSettings](#) opération pour répertorier les paramètres de votre compte.
2. Appelez l'[ListOfferings](#) opération pour répertorier les offres d'emplacements pour appareils mises à votre disposition.
3. Appelez l'[ListOfferingPromotions](#) opération pour répertorier les offres promotionnelles disponibles.

Note

Cette commande renvoie uniquement les promotions que vous n'avez pas encore achetées. Dès que vous achetez un ou plusieurs emplacements en bénéficiant d'une offre promotionnelle, cette promotion ne s'affiche plus dans les résultats.

4. Appelez l'[PurchaseOffering](#) opération pour acheter une offre.
5. Appelez l'[GetOfferingStatus](#) opération pour connaître le statut de l'offre.

Les [ListOfferingTransactions](#) commandes [RenewOffering](#) et sont également disponibles pour cette fonctionnalité.

Pour plus d'informations sur l'utilisation de l'API Device Farm, consultez [Automatiser Device Farm](#).

Annulation d'un emplacement d'appareil dans Device Farm

Vous pouvez annuler le nombre d'emplacements pour les appareils à la fois pour les tests automatisés et pour l'accès à distance. Pour obtenir des instructions, consultez l'une des sections suivantes. Le montant débité sur votre compte pour le prochain cycle de facturation sera indiqué sous le champ de la période de facturation.

Pour plus d'informations sur les emplacements pour appareils, consultez [Acheter un emplacement pour appareil dans Device Farm](#).

Annuler un emplacement pour un appareil (console)

1. Connectez-vous à la console Device Farm à l'adresse <https://console.aws.amazon.com/devicefarm>.
2. Dans le volet de navigation, choisissez Mobile Device Testing, puis Device slots.
3. Sur la page Acheter et gérer des emplacements pour appareils, vous pouvez réduire le nombre d'emplacements pour appareils à la fois pour les tests automatisés et pour l'accès à distance en diminuant la valeur sous Période de facturation suivante. Le montant débité sur votre compte pour le prochain cycle de facturation sera indiqué sous le champ de la période de facturation.
4. Choisissez Enregistrer. Une fenêtre de confirmation des modifications s'affichera. Vérifiez les informations, puis choisissez Confirmer pour terminer la transaction.

Annuler un emplacement de périphérique (AWS CLI)

Vous pouvez exécuter la `renew-offering` commande pour modifier le nombre d'appareils pour le prochain cycle de facturation.

Annuler un emplacement de terminal (API)

Appelez l'[RenewOffering](#) opération pour modifier le nombre d'appareils sur votre compte.

Concepts d'AWS Device Farm

Device Farm est un service de test d'applications que vous pouvez utiliser pour tester et interagir avec vos applications Android, iOS et Web sur de vrais téléphones et tablettes physiques hébergés par Amazon Web Services (AWS).

Cette section décrit les concepts importants de Device Farm.

- [Prise en charge des appareils dans AWS Device Farm](#)
- [Environnements de test dans AWS Device Farm](#)
- [Exécutions](#)
- [Applications](#)
- [Rapports dans AWS Device Farm](#)
- [Séances](#)

Pour plus d'informations sur les types de tests pris en charge dans Device Farm, consultez [Frameworks de test et tests intégrés dans AWS Device Farm](#).

Prise en charge des appareils dans AWS Device Farm

Les sections suivantes fournissent des informations sur la prise en charge des appareils dans Device Farm.

Rubriques

- [Appareils pris en charge](#)
- [Pools d'appareils](#)
- [Appareils privés](#)
- [Identification de l'appareil](#)
- [Fentes pour appareils](#)
- [Applications préinstallées pour appareils](#)
- [Fonctionnalités de l'appareil](#)

Appareils pris en charge

Device Farm prend en charge des centaines d'appareils Android et iOS et de combinaisons de systèmes d'exploitation uniques et populaires. La liste des appareils disponibles s'accroît à mesure que de nouveaux appareils sont lancés. Pour obtenir la liste complète des appareils, consultez la [liste interactive des appareils dans votre AWS console](#).

Pools d'appareils

Device Farm organise ses appareils en pools d'appareils que vous pouvez utiliser pour vos tests. Ces pools d'appareils contiennent des appareils connexes, tels que des appareils qui fonctionnent uniquement sur Android ou uniquement sur iOS. Device Farm propose des pools d'appareils sélectionnés, tels que ceux des meilleurs appareils. Vous pouvez également créer des pools d'appareils afin de combiner des appareils publics et privés.

Appareils privés

Les appareils privés vous permettent d'indiquer les configurations matérielles et logicielles exactes pour répondre à vos besoins en termes de tests. Certaines configurations, telles que les appareils Android rootés, peuvent être prises en charge en tant qu'appareils privés. Chaque appareil privé est un appareil physique que Device Farm déploie en votre nom dans un centre de données Amazon. Vos appareils privés sont disponibles exclusivement pour des tests manuels et automatiques. Une fois que vous avez choisi de mettre fin à votre abonnement, le matériel est retiré de notre environnement. Pour plus d'informations, consultez [Appareils privés](#) et [Appareils privés dans AWS Device Farm](#).

Identification de l'appareil

Device Farm effectue des tests sur des appareils mobiles et tablettes physiques à partir de divers appareils OEMs.

Fentes pour appareils

Les emplacements d'appareils correspondent à la simultanéité dans laquelle le nombre d'emplacements d'appareils que vous avez acheté détermine le nombre d'appareils que vous pouvez exécuter dans des tests ou des sessions d'accès à distance.

Il existe deux types d'emplacements d'appareils :

- Un emplacement d'appareil pour l'accès à distance vous permet d'exécuter des sessions d'accès à distance simultanément.

Si vous disposez d'un emplacement d'appareil pour l'accès à distance, vous pouvez uniquement exécuter des tests sur une session d'accès à distance à la fois. Si vous achetez des emplacements d'appareils de test à distance supplémentaires, vous pouvez exécuter plusieurs sessions simultanément.

- Un emplacement d'appareil de test automatisé vous permet d'exécuter des tests simultanément.

Si vous disposez d'un emplacement d'appareil de test automatisé, vous pouvez uniquement exécuter des tests sur un appareil à la fois. Si vous achetez des emplacements d'appareils de test automatisé supplémentaires, vous pouvez exécuter plusieurs tests simultanément sur plusieurs appareils afin d'obtenir des résultats de test plus rapidement.

Vous pouvez acheter des emplacements d'appareils en fonction du type d'appareil (appareils Android ou iOS pour les tests automatisés et appareils iOS et Android pour l'accès à distance). Pour plus d'informations, consultez [Tarification de Device Farm](#).

Applications préinstallées pour appareils

Les appareils de Device Farm incluent un petit nombre d'applications déjà installées par les fabricants et les opérateurs.

Fonctionnalités de l'appareil

Tous les appareils sont connectés à Internet. Ils ne disposent pas de connexion à un opérateur. Ils ne peuvent donc pas être utilisés pour passer des appels ou envoyer des SMS.

Vous pouvez prendre des photos avec n'importe quel appareil disposant d'une caméra frontale ou arrière. En raison de la manière dont les appareils sont montés, les photos peuvent être sombres ou floues.

Les services Google Play et Google Chrome sont installés sur les appareils Android.

Environnements de test dans AWS Device Farm

AWS Device Farm fournit des environnements de test personnalisés et standard pour exécuter vos tests automatisés. Vous pouvez choisir un environnement de test personnalisé pour avoir un contrôle

total sur vos tests automatisés. Vous pouvez également choisir l'environnement de test standard par défaut de Device Farm, qui propose des rapports détaillés sur chaque test de votre suite de tests automatisés.

Rubriques

- [Environnement de test standard](#)
- [Environnement de test personnalisé](#)

Environnement de test standard

Lorsque vous exécutez un test dans un environnement standard, Device Farm fournit des journaux et des rapports détaillés pour chaque cas de votre suite de tests. Vous pouvez consulter des données de performance, des vidéos, des captures d'écran et des journaux pour chaque test afin d'identifier et de résoudre les problèmes de votre application.

Note

Device Farm fournissant des rapports détaillés dans l'environnement standard, les délais d'exécution des tests peuvent être plus longs que lorsque vous les exécutez localement. Si vous souhaitez des temps d'exécution plus rapides, exécutez vos tests dans un environnement de test personnalisé.

Environnement de test personnalisé

Lorsque vous personnalisez l'environnement de test, vous pouvez spécifier les commandes que Device Farm doit exécuter pour exécuter vos tests. Cela garantit que les tests sur Device Farm s'exécutent de la même manière que les tests exécutés sur votre machine locale. L'exécution des tests dans ce mode vous permet également d'obtenir des journaux en direct et un streaming vidéo de vos tests. Lorsque vous exécutez des tests dans un environnement de test personnalisé, vous n'obtenez pas des rapports détaillés pour chaque test. Pour de plus amples informations, veuillez consulter [Environnements de test personnalisés dans AWS Device Farm](#).

Vous avez la possibilité d'utiliser un environnement de test personnalisé lorsque vous utilisez la console Device Farm ou l'API Device Farm pour créer un essai. AWS CLI

Pour plus d'informations, consultez la section [Téléchargement d'une spécification de test personnalisée à l'aide du AWS CLI](#) et [Création d'un test dans Device Farm](#)

Fonctionne dans AWS Device Farm

Les sections suivantes contiennent des informations sur les exécutions dans Device Farm.

Une exécution dans Device Farm représente une version spécifique de votre application, avec un ensemble spécifique de tests, à exécuter sur un ensemble spécifique d'appareils. Une exécution produit un rapport contenant des informations sur les résultats de l'exécution. Une exécution contient une ou plusieurs tâches.

Rubriques

- [Exécuter la configuration](#)
- [Exécuter la conservation des fichiers](#)
- [Exécuter l'état de l'appareil](#)
- [Sorties en parallèle](#)
- [Configuration du délai d'exécution](#)
- [Publicités en séries](#)
- [Médias en course](#)
- [Tâches courantes pour les courses](#)

Exécuter la configuration

Dans le cadre d'une exécution, vous pouvez fournir les paramètres que Device Farm peut utiliser pour remplacer les paramètres actuels de l'appareil. Il s'agit notamment des coordonnées de latitude et de longitude, des données supplémentaires (contenues dans un fichier .zip) et des applications auxiliaires (applications qui doivent être installées avant l'application à tester). Sur Android, certains paramètres supplémentaires peuvent être modifiés, tels que les paramètres régionaux et les états radio (Bluetooth, GPS, NFC et Wi-Fi).

Exécuter la conservation des fichiers

Device Farm stocke vos applications et vos fichiers pendant 30 jours, puis les supprime de son système. Toutefois, vous pouvez supprimer vos fichiers à tout moment.

Device Farm conserve les résultats de vos courses, vos journaux et vos captures d'écran pendant 400 jours, puis les supprime de son système.

Exécuter l'état de l'appareil

Device Farm redémarre toujours un appareil avant de le rendre disponible pour la prochaine tâche.

Sorties en parallèle

Device Farm exécute des tests en parallèle au fur et à mesure que les appareils sont disponibles.

Configuration du délai d'exécution

Vous pouvez définir une valeur de durée d'exécution des tests avant d'empêcher chaque appareil d'exécuter un test. Par exemple, si vos tests prennent 20 minutes par appareil, vous devez choisir un délai d'exécution de 30 minutes par appareil.

Pour de plus amples informations, veuillez consulter [Définition du délai d'exécution pour les tests dans AWS Device Farm](#).

Publicités en séries

Nous vous recommandons de supprimer les publicités de vos applications avant de les télécharger sur Device Farm. Nous ne pouvons pas garantir l'affichage des publicités lors des exécutions.

Médias en course

Vous pouvez fournir du multimédia ou d'autres données pour accompagner votre application. Les données supplémentaires doivent être fournies dans un fichier .zip d'une taille inférieure à 4 Go.

Tâches courantes pour les courses

Pour plus d'informations, consultez [Création d'un test dans Device Farm](#) et [Tests exécutés dans AWS Device Farm](#).

Applications dans AWS Device Farm

Les sections suivantes contiennent des informations sur le comportement des applications dans Device Farm.

Rubriques

- [Applications d'instrumentation](#)

- [Resignature d'applications en plusieurs fois](#)
- [Applications obfusquées en cours d'exécution](#)

Applications d'instrumentation

Vous n'avez pas besoin d'instrumenter vos applications ni de fournir à Device Farm le code source de vos applications. Les applications Android peuvent être envoyées non modifiées. Les applications iOS doivent être conçues pour la cible iOS Device (Appareil iOS) et non pour le simulateur.

Resignature d'applications en plusieurs fois

Pour les applications iOS, vous n'avez pas besoin d'ajouter de Device Farm UUIDs à votre profil d'approvisionnement. Device Farm remplace le profil d'approvisionnement intégré par un profil générique, puis signe à nouveau l'application. Si vous fournissez des données auxiliaires, Device Farm les ajoute au package de l'application avant que Device Farm ne l'installe, de sorte que les données auxiliaires existent dans le sandbox de votre application. La nouvelle signature de l'application supprime les droits tels que le groupe d'applications, les domaines associés, le Game Center, HealthKit, la configuration des accessoires sans fil HomeKit, les achats intégrés, l'audio entre applications, Apple Pay, les notifications push et la configuration et le contrôle du VPN.

Pour les applications Android, Device Farm signe à nouveau l'application. Cela peut interrompre toute fonctionnalité dépendant de la signature de l'application, telle que l'API Android de Google Maps, ou déclencher une détection antipiratage ou antialtération à partir de produits tels que DexGuard

Applications obfusquées en cours d'exécution

Pour les applications Android, si l'application est obfusquée, vous pouvez toujours la tester avec Device Farm si vous utilisez ProGuard. Toutefois, si vous utilisez DexGuard des mesures antipiratage, Device Farm ne peut pas resigner l'application ni effectuer de tests sur celle-ci.

Rapports dans AWS Device Farm

Les sections suivantes fournissent des informations sur les rapports de test de Device Farm.

Rubriques

- [Conservation des rapports](#)
- [Composants du rapport](#)

- [Se connecte aux rapports](#)
- [Tâches courantes pour les rapports](#)

Conservation des rapports

Device Farm conserve vos rapports pendant 400 jours. Ces rapports incluent des métadonnées, des journaux, des captures d'écran et des données de performance.

Composants du rapport

Les rapports de Device Farm contiennent des informations de réussite et d'échec, des rapports d'erreur, des journaux des tests et des appareils, des captures d'écran et des données de performance.

Les rapports incluent des données détaillées par appareil et des résultats généraux, tels que le nombre d'occurrences d'un problème donné.

Se connecte aux rapports

Les rapports incluent des captures logcat complètes pour les tests Android et des journaux de console d'appareil complets pour les tests iOS.

Tâches courantes pour les rapports

Pour de plus amples informations, veuillez consulter [Afficher les rapports de test dans Device Farm](#).

Sessions dans AWS Device Farm

Device Farm vous permet de tester de manière interactive des applications Android et iOS par le biais de sessions d'accès à distance. Cela inclut à la fois une interaction manuelle dans un navigateur Web et l'exécution de tests Appium depuis un client local sur l'appareil distant. Les développeurs peuvent reproduire les problèmes liés à leur application ou à leurs tests Appium sur un appareil spécifique afin d'isoler et de résoudre les problèmes.

Rubriques

- [Appareils compatibles pour l'accès à distance](#)
- [Conservation des fichiers de session](#)

- [Applications d'instrumentation](#)
- [Resignature d'applications dans les sessions](#)
- [Applications masquées dans les sessions](#)

Appareils compatibles pour l'accès à distance

Device Farm prend en charge un certain nombre d'appareils Android et iOS uniques et populaires. La liste des appareils disponibles s'accroît à mesure que de nouveaux appareils sont lancés. La console Device Farm affiche la liste actuelle des appareils Android et iOS disponibles pour un accès à distance. Pour de plus amples informations, veuillez consulter [Prise en charge des appareils dans AWS Device Farm](#).

Conservation des fichiers de session

Device Farm stocke vos applications et vos fichiers pendant 30 jours, puis les supprime de son système. Toutefois, vous pouvez supprimer vos fichiers à tout moment.

Device Farm conserve les journaux de vos sessions et les vidéos capturées pendant 400 jours, puis les supprime de son système.

Applications d'instrumentation

Vous n'avez pas besoin d'instrumenter vos applications ni de fournir à Device Farm le code source de vos applications. Les applications Android et iOS peuvent être envoyées non modifiées.

Resignature d'applications dans les sessions

Device Farm signe les applications Android et iOS. Cela peut interrompre les fonctionnalités reposant sur la signature de l'application. Par exemple, l'API Google Maps pour Android dépend de la signature de votre application. La nouvelle signature des applications peut également déclencher une détection antipiratage ou antialtération sur des produits tels que DexGuard les appareils Android.

Applications masquées dans les sessions

Pour les applications Android, si l'application est masquée, vous pouvez toujours la tester avec Device Farm si vous utilisez ProGuard. Toutefois, si vous utilisez l'application dans le DexGuard cadre de mesures antipiratage, Device Farm ne pourra pas signer à nouveau l'application.

Projets dans AWS Device Farm

Un projet dans Device Farm représente un espace de travail logique dans Device Farm qui contient des exécutions, une exécution pour chaque test d'une seule application sur un ou plusieurs appareils. Les projets vous permettent d'organiser les espaces de travail comme vous le souhaitez. Par exemple, il peut y avoir un projet par titre d'application ou un projet par plateforme. Vous pouvez créer autant de projets que vous en avez besoin.

Vous pouvez utiliser la console AWS Device Farm AWS Command Line Interface (AWS CLI) ou l'API AWS Device Farm pour travailler sur des projets.

Rubriques

- [Création d'un projet dans AWS Device Farm](#)
- [Afficher la liste des projets dans AWS Device Farm](#)

Création d'un projet dans AWS Device Farm

Vous pouvez créer un projet à l'aide de la console AWS Device Farm ou de l'API AWS Device Farm. AWS CLI

Conditions préalables

- Suivez les étapes de [Configuration](#).

Création d'un projet (console)

1. Connectez-vous à la console Device Farm à l'adresse <https://console.aws.amazon.com/devicefarm>.
2. Sur le panneau de navigation de Device Farm, choisissez Mobile Device Testing, puis Projects.
3. Choisissez New Project (Nouveau projet).
4. Tapez un nom pour votre projet. Vous pouvez éventuellement fournir un ou plusieurs des paramètres ci-dessous, puis choisir Soumettre.

Paramètres du Virtual Private Cloud (VPC)

Sélectionnez un VPC, des sous-réseaux et un groupe de sécurité à appliquer au périphérique testé et à son hôte de test couplé. Cette fonctionnalité n'est prise en charge que sur les appareils privés. Pour plus d'informations, consultez [VPC-ENI dans AWS Device Farm](#).

Rôle d'exécution ARN

Rôle IAM à assumer par le testeur dans les environnements de test personnalisés. Pour de plus amples informations, veuillez consulter [Accédez aux ressources AWS à l'aide d'un rôle d'exécution IAM](#).

Variables d'environnement

Une ou plusieurs variables à insérer dans l'environnement du processus d'exécution des tests. Les noms de variables commençant par « DEVICEFARM_ » sont réservés à l'utilisation du service. Nous vous déconseillons de stocker des valeurs sensibles dans ces variables d'environnement et vous suggérons plutôt d'utiliser un rôle d'exécution IAM pour récupérer ces valeurs depuis AWS Secrets Manager lors de votre test.

Création d'un projet (AWS CLI)

- Exécutez `create-project` en spécifiant le nom du projet.

Exemple :

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

La AWS CLI réponse inclut l'Amazon Resource Name (ARN) du projet.

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

Pour plus d'informations, consultez [create-project](#) et [AWS CLI référence](#).

Création d'un projet (API)

- Appelez l'API [CreateProject](#).

Pour plus d'informations sur l'utilisation de l'API Device Farm, consultez [Automatiser Device Farm](#).

Afficher la liste des projets dans AWS Device Farm

Vous pouvez utiliser la console AWS Device Farm ou l'API AWS Device Farm pour consulter la liste des projets. AWS CLI

Rubriques

- [Prérequis](#)
- [Afficher la liste des projets \(console\)](#)
- [Afficher la liste des projets \(AWS CLI\)](#)
- [Afficher la liste des projets \(API\)](#)

Prérequis

- Créez au moins un projet dans Device Farm. Suivez les instructions de [Création d'un projet dans AWS Device Farm](#), puis revenez à cette page.

Afficher la liste des projets (console)

1. Connectez-vous à la console Device Farm à l'adresse <https://console.aws.amazon.com/devicefarm>.
2. Pour trouver la liste des projets disponibles, procédez comme suit :
 - Pour les projets de test d'appareils mobiles, dans le menu de navigation de Device Farm, choisissez Mobile Device Testing, puis Projects.
 - Pour les projets de test de navigateurs de bureau, dans le menu de navigation de Device Farm, choisissez Desktop Browser Testing, puis Projects.

Afficher la liste des projets (AWS CLI)

- Pour afficher la liste des projets, exécutez la commande [list-projects](#).

Pour afficher des informations sur un seule projet, exécutez la commande [get-project](#).

Pour plus d'informations sur l'utilisation de Device Farm avec le AWS CLI, consultez [AWS CLI référence](#).

Afficher la liste des projets (API)

- Pour afficher la liste des projets, appelez l'API [ListProjects](#).

Pour afficher des informations sur un seul projet, appelez l'API [GetProject](#).

Pour plus d'informations sur l'API AWS Device Farm, consultez [Automatiser Device Farm](#).

Tests exécutés dans AWS Device Farm

Une exécution dans Device Farm représente une version spécifique de votre application, avec un ensemble spécifique de tests, à exécuter sur un ensemble spécifique d'appareils. Une exécution produit un rapport contenant des informations sur les résultats de l'exécution. Une exécution contient une ou plusieurs tâches. Pour de plus amples informations, veuillez consulter [Exécutions](#).

Vous pouvez utiliser la console AWS Device Farm, l'interface de ligne de commande AWS CLI ou l'API AWS Device Farm pour effectuer des tests.

Rubriques

- [Création d'un test dans Device Farm](#)
- [Définition du délai d'exécution pour les tests dans AWS Device Farm](#)
- [Simulation des connexions réseau et des conditions pour les exécutions de votre AWS Device Farm](#)
- [Arrêter une exécution dans AWS Device Farm](#)
- [Afficher la liste des exécutions dans AWS Device Farm](#)
- [Création d'un pool d'appareils dans AWS Device Farm](#)
- [Analyse des résultats des tests dans AWS Device Farm](#)

Création d'un test dans Device Farm

Vous pouvez utiliser la console Device Farm ou l'API Device Farm pour créer un test. AWS CLI Vous pouvez également utiliser un plugin compatible, tel que les plugins Jenkins ou Gradle pour Device Farm. Pour de plus amples informations sur les plug-ins, consultez [Outils et plugins](#). Pour plus d'informations sur les exécutions, consultez [Exécutions](#).

Rubriques

- [Conditions préalables](#)
- [Création d'un test \(console\)](#)
- [Créer un test \(AWS CLI\)](#)
- [Création d'un test \(API\)](#)
- [Étapes suivantes](#)

Conditions préalables

Vous devez avoir un projet dans Device Farm. Suivez les instructions de [Création d'un projet dans AWS Device Farm](#), puis revenez à cette page.

Création d'un test (console)

1. Connectez-vous à la console Device Farm à l'adresse <https://console.aws.amazon.com/devicefarm>.
2. Dans le volet de navigation, choisissez Mobile Device Testing, puis Projects.
3. Si vous disposez déjà d'un projet, vous pouvez charger vos tests dans celui-ci. Sinon, choisissez Nouveau projet, entrez un nom de projet, puis choisissez Créer.
4. Ouvrez votre projet, puis choisissez Create run.
5. (Facultatif) Sous Paramètres d'exécution, dans la section Nom de l'exécution, entrez le nom de votre course. Si aucun nom n'est fourni, la console Device Farm nommera votre course « My Device Farm run » par défaut.
6. (Facultatif) Sous Paramètres d'exécution, dans la section Job timeout, vous pouvez spécifier le délai d'exécution de votre test. Si vous utilisez un nombre illimité de créneaux de test, vérifiez que Unmetered est sélectionné sous Mode de facturation.
7. Sous Paramètres d'exécution, dans la section Type d'exécution, sélectionnez votre type d'exécution. Sélectionnez une application Android si aucune application n'est prête à être testée ou si vous testez une application Android (.apk). Sélectionnez une application iOS si vous testez une application iOS (.ipa). Sélectionnez Application Web si vous souhaitez tester des applications Web.
8. Sous Sélectionner une application, dans la section Options de sélection des applications, choisissez Sélectionner un exemple d'application fourni par Device Farm si aucune application n'est disponible pour les tests. Si vous apportez votre propre application, sélectionnez Télécharger votre propre application, puis choisissez votre fichier de candidature. Si vous chargez une application iOS, veillez à choisir iOS device (Appareil iOS) et non un simulateur.
9. Sous Configurer le test, choisissez l'un des frameworks de test disponibles.

Note

Si vous ne disposez d'aucun test disponible, choisissez Built-in: Fuzz (Fuzz intégré) pour exécuter une suite de tests intégrée standard. Si vous choisissez Built-in: Fuzz (Fuzz intégré) et que les zones Event count (Nombre d'événements), Event throttle (Limitation

d'événement) et Randomizer seed (Valeur initiale de générateur aléatoire) s'affichent, vous pouvez modifier ou conserver les valeurs.

Pour plus d'informations sur les suites de tests disponibles, consultez [Frameworks de test et tests intégrés dans AWS Device Farm](#).

10. Si vous n'avez pas sélectionné Built-in : Fuzz, sélectionnez Choisir un fichier sous Sélectionner un package de test. Naviguez jusqu'au fichier contenant vos tests et choisissez-le.
11. Pour votre environnement de test, choisissez Exécuter votre test dans notre environnement standard ou Exécuter votre test dans un environnement personnalisé. Pour de plus amples informations, veuillez consulter [Environnements de test dans AWS Device Farm](#).
12. Si vous utilisez un environnement de test personnalisé, vous pouvez éventuellement effectuer les opérations suivantes :
 - Si vous souhaitez modifier la spécification de test par défaut dans un environnement de test personnalisé, choisissez Edit (Modifier) pour mettre à jour la spécification YAML par défaut.
 - Si vous avez modifié la spécification de test, choisissez Enregistrer en tant que nouveau pour la mettre à jour.
 - Vous pouvez configurer des variables d'environnement. Les variables fournies ici auront priorité sur celles qui peuvent être configurées sur le projet parent.
13. Sous Sélectionner les appareils, effectuez l'une des opérations suivantes :
 - Pour choisir un groupe d'appareils intégré sur lequel exécuter les tests, pour Device pool (Groupe d'appareils), choisissez Top Devices (Principaux appareils).
 - Pour créer votre propre groupe d'appareils sur lequel exécuter les tests, suivez les instructions fournies dans [Création d'un pool d'appareils](#), puis revenez à cette page.
 - Si vous avez créé votre propre groupe d'appareils précédemment, pour Device pool (Groupe d'appareils), choisissez votre groupe d'appareils.
 - Sélectionnez Sélectionner manuellement les appareils et choisissez les appareils sur lesquels vous souhaitez vous lancer. Cette configuration ne sera pas enregistrée.

Pour de plus amples informations, veuillez consulter [Prise en charge des appareils dans AWS Device Farm](#).

14. (Facultatif) Pour ajouter une configuration supplémentaire, ouvrez le menu déroulant Configuration supplémentaire. Dans cette section, vous pouvez effectuer l'une des opérations suivantes :
- Pour fournir un ARN de rôle d'exécution ou remplacer un ARN configuré dans le projet parent, utilisez le champ ARN du rôle d'exécution.
 - Pour fournir d'autres données à utiliser par Device Farm pendant l'exécution, à côté de Ajouter des données supplémentaires, choisissez Choose File, puis naviguez jusqu'au fichier .zip contenant les données et sélectionnez-le.
 - Pour installer une application supplémentaire que Device Farm pourra utiliser pendant l'exécution, à côté de Installer d'autres applications, choisissez Choose File, puis naviguez jusqu'au fichier .apk ou .ipa qui contient l'application et sélectionnez-le. Répétez cette procédure pour les autres applications que vous voulez installer. Vous pouvez modifier l'ordre d'installation en faisant glisser et en déposant les applications après les avoir chargées.
 - Pour spécifier si le Wi-Fi, le Bluetooth, le GPS ou le NFC doivent être activés pendant l'exécution, en regard de Set radio states (Définir les états radio), cochez les cases appropriées.
 - Pour prédéfinir la latitude et la longitude de l'appareil pour l'exécution, saisissez les coordonnées en regard de Device location (Emplacement de l'appareil).
 - Pour prédéfinir les paramètres régionaux de l'appareil pour l'exécution, dans Paramètres régionaux de l'appareil, choisissez les paramètres régionaux.
 - Sélectionnez Activer l'enregistrement vidéo pour enregistrer une vidéo pendant le test.
 - Sélectionnez Activer la capture des données de performance des applications pour capturer les données de performance de l'appareil.

 Note

Le réglage de l'état radio et des paramètres régionaux de l'appareil ne sont actuellement disponibles que pour les tests natifs d'Android.

 Note

Si vous avez des appareils privés, la configuration spécifique aux appareils privés est également affichée.

15. Au bas de la page, choisissez **Create run** pour planifier l'exécution.

Device Farm démarre l'exécution dès que les appareils sont disponibles, généralement en quelques minutes. Pendant votre test, la console Device Farm affiche une icône en attente



dans le tableau des exécutions. Chaque appareil en cours d'exécution démarrera également par l'icône en attente, puis passera à l'icône en cours d'exécution



lorsque le test débutera. À la fin de chaque test, une icône de résultat de test s'affiche à côté du nom de l'appareil. Lorsque tous les tests sont terminés, l'icône en attente à côté de l'exécution devient une icône de résultat de test.

Si vous souhaitez arrêter le test, consultez [Arrêter une exécution dans AWS Device Farm](#).

Créer un test (AWS CLI)

Vous pouvez utiliser le AWS CLI pour créer un essai.

Rubriques

- [Étape 1 : Choisissez un projet](#)
- [Étape 2 : Choisissez un pool d'appareils](#)
- [Étape 3 : Téléchargez votre dossier de candidature](#)
- [Étape 4 : Téléchargez votre package de scripts de test](#)
- [Étape 5 : \(Facultatif\) Téléchargez votre spécification de test personnalisée](#)
- [Étape 6 : Planifier un test](#)

Étape 1 : Choisissez un projet

Vous devez associer votre test à un projet Device Farm.

1. Pour répertorier vos projets Device Farm, lancez `list-projects`. Si vous n'avez aucun projet, consultez [Création d'un projet dans AWS Device Farm](#).

Exemple :

```
aws devicefarm list-projects
```

La réponse inclut une liste de vos projets Device Farm.

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

2. Choisissez un projet à associer à votre exécution de test et notez son Amazon Resource Name (ARN).

Étape 2 : Choisissez un pool d'appareils

Vous devez choisir un groupe d'appareils à associer à votre exécution de test.

1. Pour afficher vos groupes d'appareils, exécutez la commande `list-device-pools` en spécifiant l'ARN de votre projet.

Exemple :

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

La réponse inclut les pools de périphériques Device Farm intégrés, tels que Top Devices, et tous les pools de périphériques créés précédemment pour ce projet :

```
{
  "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. Choisissez un groupe d'appareils et notez son ARN.

Vous pouvez également créer un groupe d'appareils, puis revenir à cette étape. Pour de plus amples informations, veuillez consulter [Création d'un pool d'appareils \(AWS CLI\)](#).

Étape 3 : Téléchargez votre dossier de candidature

Pour créer votre demande de téléchargement et obtenir une URL de téléchargement présignée Amazon Simple Storage Service (Amazon S3), vous devez :

- Votre ARN de projet.
- Le nom de votre fichier d'application
- Le type de chargement

Pour de plus amples informations, veuillez consulter [create-upload](#).

1. Pour charger un fichier, exécutez la commande `create-upload` avec les paramètres `--project-arn`, `--name` et `--type`.

Cet exemple crée un chargement pour une application Android :

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

La réponse inclut l'ARN de chargement de l'application et une URL présignée.

```
{
  "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. Notez l'ARN de chargement de l'application et l'URL présignée.
3. Chargez le fichier de votre application à l'aide de l'URL présignée Amazon S3. Cet exemple utilise `curl` pour charger un fichier `.apk` Android :

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

Pour plus d'informations, consultez la section [Chargement d'objets à l'aide de Presigned URLs](#) dans le guide de l'utilisateur d'Amazon Simple Storage Service.

4. Pour vérifier l'état de votre chargement d'application, exécutez `get-upload` et spécifiez l'ARN de chargement de l'application.

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

Attendez que l'état de la réponse soit SUCCEEDED avant de charger votre package de scripts de test.

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

Étape 4 : Téléchargez votre package de scripts de test

Ensuite, chargez votre package de scripts de test.

1. Pour créer votre demande de téléchargement et obtenir une URL de téléchargement présignée Amazon S3, exécutez `create-upload` avec les `--type` paramètres `--project-arn` `--name`, et.

Cet exemple crée un chargement de package de test Appium Java TestNG :

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

La réponse inclut l'ARN de chargement de votre package de test et une URL présignée.

```
{
  "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. Notez l'ARN de chargement du package de test et l'URL présignée.
3. Téléchargez le fichier de package de vos scripts de test à l'aide de l'URL présignée Amazon S3. Cet exemple utilise curl pour charger un fichier compressé de scripts Appium TestNG :

```

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

```

4. Pour vérifier l'état du chargement de votre package de scripts de test, exécutez get-upload et spécifiez l'ARN de chargement du package de test, obtenu à l'étape 1.

```

aws devicefarm get-upload --arn arn:MyTestsUploadARN

```

Attendez que l'état de la réponse soit SUCCEEDED avant de passer à l'étape suivante, facultative.

```

{
  "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}"
  }
}

```

Étape 5 : (Facultatif) Téléchargez votre spécification de test personnalisée

Si vous exécutez vos tests dans un environnement de test standard, ignorez cette étape.

Device Farm gère un fichier de spécifications de test par défaut pour chaque type de test pris en charge. Ensuite, téléchargez votre spécification de test par défaut et utilisez-la pour créer un

chargement de spécification de test personnalisée afin d'exécuter vos tests dans un environnement de test personnalisé. Pour de plus amples informations, veuillez consulter [Environnements de test dans AWS Device Farm](#).

1. Pour trouver l'ARN de chargement de votre spécification de test par défaut, exécutez la commande `list-uploads` et spécifiez l'ARN de votre projet.

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

La réponse contient une entrée pour chaque spécification de test par défaut :

```
{
  "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. Choisissez votre spécification de test par défaut dans la liste. Notez son ARN de chargement.
3. Pour télécharger votre spécification de test par défaut, exécutez la commande `get-upload` et spécifiez l'ARN de chargement.

Exemple :

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

La réponse contient une URL présignée à laquelle vous pouvez télécharger votre spécification de test par défaut.

4. Cet exemple utilise curl pour télécharger la spécification de test par défaut et l'enregistrer sous `MyTestSpec.yml` :

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

5. Vous pouvez modifier la spécification de test par défaut pour qu'elle réponde à vos exigences en matière de test, puis utiliser votre spécification de test modifiée lors des futures exécutions de test. Ignorez cette étape pour utiliser la spécification de test par défaut en l'état dans un environnement de test personnalisé.
6. Pour créer un chargement de votre spécification de test personnalisée, exécutez la commande `create-upload` en spécifiant le nom de votre spécification de test, le type de spécification de test et l'ARN du projet.

Cet exemple crée un chargement pour une spécification de test personnalisée Appium Java TestNG :

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

La réponse inclut l'ARN de chargement de la spécification de test et une URL présignée :

```
{
  "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. Notez l'ARN de chargement de la spécification de test et l'URL présignée.
8. Téléchargez votre fichier de spécifications de test à l'aide de l'URL présignée Amazon S3. Cet exemple permet curl de télécharger une spécification de test Appium JavaTest NG :


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

9. Pour vérifier l'état de votre chargement de spécification de test, exécutez `get-upload` et spécifiez l'ARN de chargement.

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

Attendez que l'état de la réponse soit `SUCCEEDED` avant de planifier l'exécution de votre test.

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

Pour mettre à jour votre spécification de test personnalisée, exécutez la commande `update-upload` en spécifiant l'ARN de chargement de la spécification de test. Pour de plus amples informations, veuillez consulter [update-upload](#).

Étape 6 : Planifier un test

Pour planifier un test avec le AWS CLI `schedule-run`, exécutez en spécifiant :

- L'ARN du projet, obtenu à l'[étape 1](#)
- L'ARN du groupe d'appareils, obtenu à l'[étape 2](#)
- L'ARN de chargement de l'application, obtenu à l'[étape 3](#)
- L'ARN de chargement du package de test, obtenu à l'[étape 4](#)

Si vous exécutez des tests dans un environnement de test personnalisé, vous avez également besoin de l'ARN de votre spécification de test, obtenu à l'[étape 5](#).

Pour planifier une exécution dans un environnement de test standard

- Exécutez la commande `schedule-run` en spécifiant l'ARN de votre projet, l'ARN du groupe d'appareils, l'ARN de chargement de l'application et les informations sur le package de test.

Exemple :

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

La réponse contient un ARN d'exécution que vous pouvez utiliser pour vérifier l'état de votre exécution de test.

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

```

    "arn": "arn:aws:devicefarm:us-west-2:123456789101:run:5e01a8c7-c861-4c0a-b1d5-12345runEXAMPLE",
    "counters": {
      "skipped": 0,
      "warned": 0,
      "failed": 0,
      "stopped": 0,
      "passed": 0,
      "errored": 0,
      "total": 0
    }
  }
}

```

Pour de plus amples informations, veuillez consulter [schedule-run](#).

Pour planifier une exécution dans un environnement de test personnalisé

- Les étapes sont presque identiques à celles à suivre pour l'environnement de test standard, avec un attribut supplémentaire `testSpecArn` inclus dans le paramètre `--test`.

Exemple :

```

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

```

Pour vérifier l'état de votre exécution de test

- Utilisez la commande `get-run` et spécifiez l'ARN d'exécution :

```

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

```

Pour de plus amples informations, veuillez consulter [get-run](#). Pour plus d'informations sur l'utilisation de Device Farm avec le AWS CLI, consultez [AWS CLI référence](#).

Création d'un test (API)

Les étapes sont les mêmes que celles décrites dans la AWS CLI section. Consultez [Créer un test \(AWS CLI\)](#).

Vous avez besoin des informations suivantes pour appeler l'API [ScheduleRun](#) :

- L'ARN d'un projet. Consultez [Création d'un projet \(API\)](#) et [CreateProject](#).
- L'ARN de chargement d'une application. Consultez [CreateUpload](#).
- L'ARN de chargement d'un package de test. Consultez [CreateUpload](#).
- L'ARN d'un groupe d'appareils. Consultez [Création d'un pool d'appareils](#) et [CreateDevicePool](#).

Note

Si vous exécutez des tests dans un environnement de test personnalisé, vous avez également besoin de l'ARN de chargement de votre spécification de test. Pour plus d'informations, consultez [Étape 5 : \(Facultatif\) Téléchargez votre spécification de test personnalisée](#) et [CreateUpload](#).

Pour plus d'informations sur l'utilisation de l'API Device Farm, consultez [Automatiser Device Farm](#).

Étapes suivantes

Dans la console Device Farm, l'icône de l'horloge



devient une icône de résultat, telle que celle de succès une



fois l'exécution terminée. Un rapport sur l'exécution s'affiche dès que les tests sont terminés. Pour de plus amples informations, veuillez consulter [Rapports dans AWS Device Farm](#).

Pour utiliser ce rapport, suivez les instructions fournies dans [Afficher les rapports de test dans Device Farm](#).

Définition du délai d'exécution pour les tests dans AWS Device Farm

Vous pouvez définir une valeur de durée d'exécution des tests avant d'empêcher chaque appareil d'exécuter un test. Le délai d'exécution par défaut est de 150 minutes par appareil, mais vous pouvez définir une valeur de seulement 5 minutes. Vous pouvez utiliser la console AWS Device Farm ou AWS CLI l'API AWS Device Farm pour définir le délai d'exécution.

Important

L'option de délai d'exécution doit être définie sur la durée maximale d'une exécution de test, plus un tampon. Par exemple, si vos tests prennent 20 minutes par appareil, vous devez choisir un délai d'exécution de 30 minutes par appareil.

Si l'exécution dépasse le délai, elle est arrêtée de force sur l'appareil concerné. Des résultats partiels sont disponibles, si possible. Vous êtes facturé à hauteur de la durée d'exécution si vous avez choisi le mode de facturation limité. Pour plus d'informations sur les tarifs, consultez [Device Farm Pricing](#).

Vous souhaitez peut-être utiliser cette fonction si vous savez combien de temps l'exécution de test doit prendre sur chaque appareil. La définition d'un délai d'exécution pour une exécution de test vous permet d'éviter de vous retrouver avec une exécution de test bloquée pour une raison quelconque et d'être facturé pour des minutes passées sur l'appareil pendant lesquelles aucun test n'est exécuté. Autrement dit, la fonction de délai d'exécution vous permet d'arrêter l'exécution si elle prend plus de temps que prévu.

Vous pouvez définir le délai d'exécution à deux niveaux : celui du projet et celui de l'exécution de test.

Conditions préalables

1. Suivez les étapes de [Configuration](#).
2. Créez un projet dans Device Farm. Suivez les instructions de [Création d'un projet dans AWS Device Farm](#), puis revenez à cette page.

Définir le délai d'exécution d'un projet

1. Connectez-vous à la console Device Farm à l'adresse <https://console.aws.amazon.com/devicefarm>.
2. Sur le panneau de navigation de Device Farm, choisissez Mobile Device Testing, puis Projects.
3. Si vous avez déjà un projet, sélectionnez-le dans la liste. Sinon, choisissez Nouveau projet, entrez un nom pour votre projet, puis choisissez Soumettre.
4. Choisissez Project settings (Paramètres du projet).
5. Sous l'onglet General (Général), pour Execution timeout (Délai d'exécution), entrez une valeur ou utilisez la barre du curseur.
6. Choisissez Enregistrer.

Toutes les exécutions de test de votre projet utilisent désormais la valeur de délai d'exécution que vous avez spécifiée, sauf si vous la remplacez lorsque vous planifiez une exécution.

Définissez le délai d'exécution d'un test

1. Connectez-vous à la console Device Farm à l'adresse <https://console.aws.amazon.com/devicefarm>.
2. Sur le panneau de navigation de Device Farm, choisissez Mobile Device Testing, puis Projects.
3. Si vous avez déjà un projet, sélectionnez-le dans la liste. Sinon, choisissez Nouveau projet, entrez un nom pour votre projet, puis choisissez Soumettre.
4. Choisissez Create a new run (Créer une exécution).
5. Suivez les étapes ci-dessous pour choisir une application, configurer votre test, sélectionner vos appareils et spécifier un état d'appareil.
6. Dans Révision et lancement de l'exécution, pour Définir le délai d'exécution, entrez une valeur ou utilisez le curseur.
7. Choisissez Confirm and start run (Confirmer et démarrer l'exécution).

Simulation des connexions réseau et des conditions pour les exécutions de votre AWS Device Farm

Vous pouvez utiliser le modelage du réseau pour simuler les connexions et les conditions du réseau lorsque vous testez vos applications Android, iOS et Web dans Device Farm. Par exemple, vous pouvez simuler une connexion Internet intermittente ou avec perte.

Lorsque vous créez une exécution en utilisant les paramètres réseau par défaut, chaque appareil dispose d'une connexion Wi-Fi complète, intègre, avec une connectivité Internet. Lorsque vous utilisez le modelage du réseau, vous pouvez modifier la connexion Wi-Fi pour spécifier un profil réseau tel que 3G ou Lossy WiFi qui contrôle le débit, le délai, l'instabilité et les pertes pour le trafic entrant et sortant.

Rubriques

- [Configurer le modelage du réseau lors de la planification d'un test](#)
- [Création d'un profil réseau](#)
- [Modifier les conditions du réseau pendant votre test](#)

Configurer le modelage du réseau lors de la planification d'un test

Lorsque vous planifiez une course, vous pouvez choisir l'un des profils sélectionnés par Device Farm, ou vous pouvez créer et gérer le vôtre.

1. Dans n'importe quel projet Device Farm, choisissez Create a new run.

Si vous n'avez pas encore de projet, consultez [Création d'un projet dans AWS Device Farm](#).

2. Choisissez votre application, puis cliquez sur Suivant.
3. Configurez votre test, puis choisissez Next.
4. Sélectionnez vos appareils, puis cliquez sur Suivant.
5. Dans la section Emplacement et paramètres réseau, choisissez un profil réseau ou choisissez Créer un profil réseau pour créer le vôtre.

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. Choisissez Suivant.
7. Vérifiez et démarrez votre exécution de test.

Création d'un profil réseau

Lorsque vous créez une exécution de test, vous pouvez créer un profil réseau.

1. Choisissez Créer un profil réseau.

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. Entrez le nom et les paramètres de votre profil réseau.
3. Choisissez Créer.
4. Terminez la création de votre exécution de test et démarrez l'exécution.

Après avoir créé un profil réseau, vous pourrez le voir et le gérer sur la page Project settings (Paramètres du projet).

General Device pools Network profiles Uploads						
Network profiles						
   						
	Name	Bandwidth (bps)	Delay (ms)	Jitter (ms)	Loss (%)	Description
☐		 104857600  1048576	 0  0	 0  0	 0  0	-
☐		 104857600  1048576	 0  0	 0  0	 0  0	-
☐		 104857600  1048576	 0  0	 0  0	 0  0	-

Modifier les conditions du réseau pendant votre test

Vous pouvez appeler une API depuis l'hôte de votre appareil à l'aide d'un framework tel qu'Appium pour simuler des conditions de réseau dynamiques telles qu'une réduction de la bande passante pendant votre essai. Pour de plus amples informations, veuillez consulter [CreateNetworkProfile](#).

Arrêter une exécution dans AWS Device Farm

Vous voudrez peut-être arrêter une exécution après l'avoir démarrée. Par exemple, si vous remarquez un problème pendant que vos tests s'exécutent, vous voudrez peut-être redémarrer l'exécution avec un script de test mis à jour.

Vous pouvez utiliser la console Device Farm ou l'API pour arrêter une exécution. AWS CLI

Rubriques

- [Arrêter une exécution \(console\)](#)
- [Arrêtez une course \(AWS CLI\)](#)
- [Arrêter une exécution \(API\)](#)

Arrêter une exécution (console)

1. Connectez-vous à la console Device Farm à l'adresse <https://console.aws.amazon.com/devicefarm>.
2. Sur le panneau de navigation de Device Farm, choisissez Mobile Device Testing, puis Projects.
3. Choisissez le projet pour lequel vous effectuez un test actif.
4. Sur la page Tests automatisés, choisissez l'exécution du test.

L'icône en attente ou en cours d'exécution doit apparaître à gauche du nom de l'appareil.

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

Run ARN:   Stop run

No recent tests

Passed Failed Errored Warned Stopped Skipped

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

0 out of 5 devices completed 0%

Devices Unique problems Screenshots Parsing result

Devices

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

5. Choisissez Stop run (Arrêter l'exécution).

Après un court instant, une icône avec un cercle rouge avec un signe négatif à l'intérieur apparaît à côté du nom de l'appareil. Lorsque la course est arrêtée, la couleur de l'icône passe du rouge au noir.

Important

Si un test a déjà été effectué, Device Farm ne peut pas l'arrêter. Si un test est en cours, Device Farm l'arrête. Le nombre total de minutes qui vous sera facturé s'affiche dans la section Devices (Appareils). En outre, vous serez également facturé pour le nombre total de minutes nécessaires à Device Farm pour exécuter la suite de configuration et la suite de démontage. Pour plus d'informations, consultez [Tarification de Device Farm](#).

L'image suivante montre un exemple de la section Devices (Appareils) après l'arrêt réussi d'une exécution de test.

Devices

Unique problems

Screenshots

Parsing result

Devices

Q

Find device by status, device name, or OS

<

1

>

⊙

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

Arrêtez une course (AWS CLI)

Vous pouvez exécuter la commande suivante pour arrêter le test spécifié, où *myARN* est le nom de ressource Amazon (ARN) du test.

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

Vous devez voir des résultats similaires à ce qui suit :

```
{
  "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
    }
}
```

Pour obtenir l'ARN de votre exécution, utilisez la commande `list-runs`. La sortie doit ressembler à ce qui suit :

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

Pour plus d'informations sur l'utilisation de Device Farm avec le AWS CLI, consultez [AWS CLI référence](#).

Arrêter une exécution (API)

- Appelez l'[StopRun](#) opération au test.

Pour plus d'informations sur l'utilisation de l'API Device Farm, consultez [Automatiser Device Farm](#).

Afficher la liste des exécutions dans AWS Device Farm

Vous pouvez utiliser la console Device Farm ou l'API pour afficher la liste des exécutions d'un projet.
AWS CLI

Rubriques

- [Afficher la liste des exécutions \(console\)](#)
- [Afficher la liste des courses \(AWS CLI\)](#)
- [Afficher la liste des exécutions \(API\)](#)

Afficher la liste des exécutions (console)

1. Connectez-vous à la console Device Farm à l'adresse <https://console.aws.amazon.com/devicefarm>.
2. Sur le panneau de navigation de Device Farm, choisissez Mobile Device Testing, puis Projects.
3. Dans la liste des projets, choisissez le projet correspondant à la liste d'exécutions que vous souhaitez afficher.

Tip

Vous pouvez utiliser la barre de recherche pour filtrer la liste des projets par nom.

Afficher la liste des courses (AWS CLI)

- Exécutez la commande [list-runs](#).

Pour afficher des informations sur une seule exécution, exécutez la commande [get-run](#).

Pour plus d'informations sur l'utilisation de Device Farm avec le AWS CLI, consultez [AWS CLI référence](#).

Afficher la liste des exécutions (API)

- Appelez l'API [ListRuns](#).

Pour afficher des informations sur une seule exécution, appelez l'API [GetRun](#).

Pour plus d'informations sur l'API Device Farm, consultez [Automatiser Device Farm](#).

Création d'un pool d'appareils dans AWS Device Farm

Vous pouvez utiliser la console Device Farm ou AWS CLI l'API pour créer un pool d'appareils.

Rubriques

- [Conditions préalables](#)
- [Création d'un pool d'appareils \(console\)](#)
- [Création d'un pool d'appareils \(AWS CLI\)](#)
- [Création d'un pool d'appareils \(API\)](#)

Conditions préalables

- Créez une exécution dans la console Device Farm. Suivez les instructions de la section [Création d'un test dans Device Farm](#). Lorsque vous accédez à la page Select devices (Sélectionner les appareils), suivez les instructions fournies dans cette section.

Création d'un pool d'appareils (console)

1. Sur la page Projets, choisissez votre projet. Sur la page Détails du projet, sélectionnez Paramètres du projet. Dans l'onglet Device pools, choisissez Create Device Pool.
2. Pour Name (Nom), saisissez un nom permettant d'identifier facilement ce groupe d'appareils.
3. Pour Description, saisissez une description permettant d'identifier facilement ce groupe d'appareils.

4. Si vous souhaitez utiliser un ou plusieurs critères de sélection pour les appareils de ce groupe, procédez comme suit :
 - a. Choisissez Créer un pool de périphériques dynamique.
 - b. Choisissez Ajouter une règle.
 - c. Pour Champ (première liste déroulante), sélectionnez l'une des options suivantes :
 - Pour inclure les appareils par leur nom de fabricant, sélectionnez Device Manufacturer.
 - Pour inclure les appareils en fonction de leur format (tablette ou téléphone), choisissez Form Factor.
 - Pour inclure les appareils en fonction de leur état de disponibilité en fonction de la charge, sélectionnez Disponibilité.
 - Pour inclure uniquement les appareils publics ou privés, choisissez le type de flotte.
 - Pour inclure les appareils en fonction de leur système d'exploitation, choisissez Platform.
 - Certains appareils sont dotés d'une étiquette ou d'une description supplémentaire à propos de l'appareil. Vous pouvez rechercher les appareils en fonction du contenu de leur étiquette en choisissant Étiquettes d'instance.
 - Pour inclure les appareils en fonction de leur version du système d'exploitation, choisissez Version du système d'exploitation.
 - Pour inclure les appareils en fonction de leur modèle, sélectionnez Modèle.
 - d. Pour Opérateur (deuxième liste déroulante), choisissez une opération logique (EQUALS, CONTAINS, etc.) pour inclure les appareils en fonction de la requête. Par exemple, vous pouvez choisir *Availability EQUALS AVAILABLE* d'inclure les appareils qui ont actuellement le Available statut.
 - e. Pour Valeur (troisième liste déroulante), entrez ou choisissez la valeur que vous souhaitez spécifier pour les valeurs de champ et d'opérateur. Les valeurs sont limitées en fonction de votre choix de champ. Par exemple, si vous choisissez Platform for Field, les seules sélections disponibles sont ANDROID et IOS. De même, si vous choisissez Form Factor pour Field, les seules sélections disponibles sont TÉLÉPHONE et TABLETTE.
 - f. Pour ajouter une autre règle, choisissez Ajouter une règle.

Une fois que vous avez créé la première règle, dans la liste des appareils, la case en regard de chaque appareil correspondant à cette règle est activée. Une fois que vous avez créé ou modifié des règles, dans la liste des appareils, la case en regard de chaque appareil correspondant à ces règles combinées est activée. Les appareils dont les cases

sont activées sont inclus dans le groupe d'appareils. Les appareils dont les cases sont désactivées sont exclus.

- g. Sous Nombre maximal d'appareils, entrez le nombre d'appareils que vous souhaitez utiliser dans votre pool d'appareils. Si vous ne saisissez pas le nombre maximum d'appareils, Device Farm sélectionnera tous les appareils du parc qui correspondent aux règles que vous avez créées. Pour éviter des frais supplémentaires, définissez ce chiffre sur un montant correspondant à vos exigences réelles en matière d'exécution parallèle et de variété d'appareils.
 - h. Pour supprimer une règle, choisissez Supprimer la règle.
5. Si vous souhaitez inclure ou exclure manuellement des appareils individuels, procédez comme suit :
 - a. Choisissez Créer un pool de périphériques statique.
 - b. Cochez ou décochez la case à côté de chaque appareil. Vous ne pouvez activer ou désactiver les cases que si vous n'avez aucune règle spécifiée.
6. Si vous souhaitez inclure ou exclure tous les appareils affichés, activez ou désactivez la case dans la ligne d'en-tête de colonne de la liste. Si vous souhaitez afficher uniquement les instances d'appareils privés, choisissez Voir uniquement les instances d'appareils privés.

Important

Bien que vous puissiez utiliser les cases de la ligne d'en-tête de colonne pour modifier la liste des appareils affichés, cela ne signifie pas que les autres appareils affichés sont les seuls à être inclus ou exclus. Pour vérifier quels appareils sont inclus ou exclus, veuillez à effacer le contenu de toutes les cases dans la ligne d'en-tête de colonne, puis à parcourir les cases.

7. Choisissez Créer.

Création d'un pool d'appareils (AWS CLI)

Tip

Si vous ne saisissez pas le nombre maximum d'appareils, Device Farm sélectionnera tous les appareils du parc qui correspondent aux règles que vous avez créées. Pour éviter des frais

supplémentaires, définissez ce chiffre sur un montant correspondant à vos exigences réelles en matière d'exécution parallèle et de variété d'appareils.

- Exécutez la commande [create-device-pool](#).

Pour plus d'informations sur l'utilisation de Device Farm avec le AWS CLI, consultez [AWS CLI référence](#).

Création d'un pool d'appareils (API)

Tip

Si vous ne saisissez pas le nombre maximum d'appareils, Device Farm sélectionnera tous les appareils du parc qui correspondent aux règles que vous avez créées. Pour éviter des frais supplémentaires, définissez ce chiffre sur un montant correspondant à vos exigences réelles en matière d'exécution parallèle et de variété d'appareils.

- Appelez l'API [CreateDevicePool](#).

Pour plus d'informations sur l'utilisation de l'API Device Farm, consultez [Automatiser Device Farm](#).

Analyse des résultats des tests dans AWS Device Farm

Dans l'environnement de test standard, vous pouvez utiliser la console Device Farm pour consulter les rapports de chaque test de votre cycle de test. La consultation des rapports vous permet de comprendre quels tests ont été réussis ou ont échoué, et vous fournit des informations détaillées sur les performances et le comportement de votre application sur les différentes configurations d'appareils.

Device Farm rassemble également d'autres artefacts tels que des fichiers, des journaux et des images que vous pouvez télécharger une fois votre test terminé. Ces informations peuvent vous aider à analyser le comportement de votre application sur de vrais appareils, à identifier les problèmes ou les bogues et à diagnostiquer les problèmes.

Rubriques

- [Afficher les rapports de test dans Device Farm](#)
- [Téléchargement d'artefacts dans Device Farm](#)

Afficher les rapports de test dans Device Farm

Utilisez la console Device Farm pour consulter vos rapports de test. Pour de plus amples informations, veuillez consulter [Rapports dans AWS Device Farm](#).

Rubriques

- [Conditions préalables](#)
- [Afficher les rapports](#)
- [Statuts des résultats des tests Device Farm](#)

Conditions préalables

Configurez une exécution de test et vérifiez qu'elle est terminée.

1. Pour créer une exécution, consultez [Création d'un test dans Device Farm](#), puis revenez à cette page.
2. Vérifiez que l'exécution est terminée. Pendant votre test, la console Device Farm affiche une icône en attente



pour les exécutions en cours. Chaque appareil en cours d'exécution commencera également par l'icône en attente, puis passera à



en cours d'exécution lorsque le test débutera. À la fin de chaque test, une icône de résultat de test s'affiche à côté du nom de l'appareil. Lorsque tous les tests sont terminés, l'icône en attente à côté de l'exécution devient une icône de résultat de test. Pour de plus amples informations, veuillez consulter [Statuts des résultats des tests Device Farm](#).

icône

Afficher les rapports

Vous pouvez consulter les résultats de votre test dans la console Device Farm.

Rubriques

- [Afficher la page récapitulative du test](#)
- [Afficher des rapports de problèmes uniques](#)
- [Afficher les rapports sur les appareils](#)
- [Afficher les rapports de la suite de tests](#)
- [Affichage des rapports de test](#)
- [Afficher les informations du journal relatives à un problème, à un appareil, à une suite ou à un test dans un rapport](#)

Afficher la page récapitulative du test

1. Connectez-vous à la console Device Farm à l'adresse <https://console.aws.amazon.com/devicefarm>.
2. Dans le volet de navigation, choisissez Mobile Device Testing, puis Projects.
3. Dans la liste des projets, choisissez le projet pour l'exécution.

Tip

Pour filtrer la liste des projets par nom, utilisez la barre de recherche.

4. Choisissez une exécution terminée pour afficher la page de son rapport récapitulatif.
5. La page récapitulative de l'exécution de test affiche une présentation de vos résultats de test.
 - La section Unique problems (Problèmes uniques) répertorie les avertissements et échecs uniques. Pour afficher les problèmes uniques, suivez les instructions fournies dans [Afficher des rapports de problèmes uniques](#).
 - La section Devices (Appareils) affiche le nombre total de tests, par résultat, pour chaque appareil.

Devices

Unique problems

Screenshots

Parsing result

Devices

Find device by status, device name, or OS

<1>

⚙

Status	Device	OS	Test Results	Total Minutes
✔ Passed	Google Pixel 4 XL (Unlocked)	10	Passed: 3, errored: 0, failed: 0	00:02:36
✔ Passed	Samsung Galaxy S20 (Unlocked)	10	Passed: 3, errored: 0, failed: 0	00:02:34
✖ Failed	Samsung Galaxy S20 ULTRA (Unlocked)	10	Passed: 2, errored: 0, failed: 1	00:02:25
✔ Passed	Samsung Galaxy S9 (Unlocked)	9	Passed: 3, errored: 0, failed: 0	00:02:46
✔ Passed	Samsung Galaxy Tab S4	8.1.0	Passed: 3, errored: 0, failed: 0	00:03:13

Dans cet exemple, il existe plusieurs appareils. Dans la première entrée du tableau, l'appareil Google Pixel 4 XL exécutant la version 10 d'Android signale trois tests réussis dont l'exécution a duré 02:36 minutes.

Pour afficher les résultats par appareil, suivez les instructions fournies dans [Afficher les rapports sur les appareils](#).

- La section Captures d'écran affiche la liste de toutes les captures d'écran capturées par Device Farm pendant l'exécution, regroupées par appareil.
- Dans la section Résultat de l'analyse, vous pouvez télécharger le résultat de l'analyse.

Afficher des rapports de problèmes uniques

1. Dans Unique problems (Problèmes uniques), choisissez le problème que vous souhaitez afficher.
2. Choisissez l'appareil. Le rapport affiche des informations sur le problème.

La section Video (Vidéo) affiche un enregistrement vidéo téléchargeable du test.

La section Résultat affiche le résultat du test. Le statut est représenté par une icône de résultat. Pour de plus amples informations, veuillez consulter [Statuts d'un test individuel](#).

La section Logs affiche toutes les informations enregistrées par Device Farm pendant le test. Pour afficher ces informations, suivez les instructions fournies dans [Afficher les informations du journal relatives à un problème, à un appareil, à une suite ou à un test dans un rapport](#).

L'onglet Fichiers affiche la liste de tous les fichiers associés au test (tels que les fichiers journaux) que vous pouvez télécharger. Pour télécharger un fichier, cliquez sur le lien du fichier dans la liste.

L'onglet Captures d'écran affiche la liste de toutes les captures d'écran capturées par Device Farm pendant le test.

Afficher les rapports sur les appareils

- Dans la section Devices (Appareils), choisissez l'appareil.

La section Video (Vidéo) affiche un enregistrement vidéo téléchargeable du test.

La section Suites affiche un tableau contenant des informations sur les suites de l'appareil.

Dans ce tableau, la colonne Résultats des tests récapitule le nombre de tests par résultat pour chacune des suites de tests exécutées sur le périphérique. Ces données comportent également une composante graphique. Pour de plus amples informations, veuillez consulter [Statuts pour plusieurs tests](#).

Pour afficher les résultats complets par suite, suivez les instructions figurant dans [Afficher les rapports de la suite de tests](#).

La section Logs affiche toutes les informations que Device Farm a enregistrées pour l'appareil pendant l'exécution. Pour afficher ces informations, suivez les instructions fournies dans [Afficher les informations du journal relatives à un problème, à un appareil, à une suite ou à un test dans un rapport](#).

La section Fichiers affiche la liste des suites pour l'appareil et tous les fichiers associés (tels que les fichiers journaux) que vous pouvez télécharger. Pour télécharger un fichier, cliquez sur le lien du fichier dans la liste.

La section Captures d'écran affiche la liste de toutes les captures d'écran capturées par Device Farm pendant l'exécution de l'appareil, regroupées par suite.

Afficher les rapports de la suite de tests

1. Dans la section Devices (Appareils), choisissez l'appareil.
2. Dans la section Suites, choisissez la suite dans le tableau.

La section Video (Vidéo) affiche un enregistrement vidéo téléchargeable du test.

La section Tests affiche un tableau contenant des informations sur les tests de la suite.

Dans le tableau, la colonne Résultats du test affiche le résultat. Ces données comportent également une composante graphique. Pour de plus amples informations, veuillez consulter [Statuts pour plusieurs tests](#).

Pour afficher les résultats complets par test, suivez les instructions de [Affichage des rapports de test](#).

La section Logs affiche toutes les informations enregistrées par Device Farm lors de l'exécution de la suite. Pour afficher ces informations, suivez les instructions fournies dans [Afficher les informations du journal relatives à un problème, à un appareil, à une suite ou à un test dans un rapport](#).

La section Fichiers affiche la liste des tests pour la suite et tous les fichiers associés (tels que les fichiers journaux) que vous pouvez télécharger. Pour télécharger un fichier, cliquez sur le lien du fichier dans la liste.

La section Captures d'écran affiche une liste de toutes les captures d'écran capturées par Device Farm lors de l'exécution de la suite, regroupées par test.

Affichage des rapports de test

1. Dans la section Devices (Appareils), choisissez l'appareil.
2. Dans la section Suites, choisissez la suite.
3. Dans la section Tests, choisissez le test.
4. La section Video (Vidéo) affiche un enregistrement vidéo téléchargeable du test.

La section Résultat affiche le résultat du test. Le statut est représenté par une icône de résultat. Pour de plus amples informations, veuillez consulter [Statuts d'un test individuel](#).

La section Logs affiche toutes les informations enregistrées par Device Farm pendant le test. Pour afficher ces informations, suivez les instructions fournies dans [Afficher les informations du journal relatives à un problème, à un appareil, à une suite ou à un test dans un rapport](#).

L'onglet Fichiers affiche la liste de tous les fichiers associés au test (tels que les fichiers journaux) que vous pouvez télécharger. Pour télécharger un fichier, cliquez sur le lien du fichier dans la liste.

L'onglet Captures d'écran affiche la liste de toutes les captures d'écran capturées par Device Farm pendant le test.

Afficher les informations du journal relatives à un problème, à un appareil, à une suite ou à un test dans un rapport

La section Logs affiche les informations suivantes :

- Source représente la source d'une entrée de journal. Les valeurs possibles incluent :
 - Harness représente une entrée de journal créée par Device Farm. Ces entrées de journal sont généralement créées lors du démarrage et de l'arrêt d'événements.
 - Le périphérique représente une entrée de journal créée par le périphérique. Pour Android, ces entrées de journal sont compatibles avec logcat. Pour iOS, ces entrées de journal sont compatibles avec syslog.
 - Test représente une entrée de journal créée par un test ou son framework de test.
- Time (Temps) représente le temps écoulé entre la première entrée de journal et cette entrée de journal. Le temps est exprimé sous **MM:SS.SSS** forme de **M** minutes et **S** de secondes.
- PID représente l'identifiant de processus (PID) ayant créé l'entrée de journal. Toutes les entrées de journal créées par une application sur un appareil ont le même PID.
- Level (Niveau) représente le niveau de journalisation de l'entrée de journal. Par exemple, `Logger.debug("This is a message!")` consigne un Level (Niveau) Debug. Voici les valeurs possibles :
 - Alert (Alerte)
 - Critical (Critique)
 - Debug (Débogage)
 - Emergency (Urgence)
 - Error (Erreur)

- Errored (Erroné)
 - Échec
 - Info (Infos)
 - Internal (Interne)
 - Notice (Avis)
 - Passed (Réussite)
 - Ignoré
 - Arrêté(e)
 - Verbose (Détailé)
 - Warned (Averti)
 - Warning (Avertissement)
- Tag (Balise) représente les métadonnées arbitraires de l'entrée de journal. Par exemple, Android logcat peut s'en servir pour décrire la partie du système ayant créé l'entrée de journal (par exemple, `ActivityManager`).
 - Message représente le message ou les données de l'entrée de journal. Par exemple, `Logger.debug("Hello, World!")` journalise le Message "Hello, World!".

Pour afficher uniquement une partie des informations :

- Pour afficher toutes les entrées du journal correspondant à la valeur d'une colonne spécifique, entrez la valeur dans la barre de recherche. Par exemple, pour afficher toutes les entrées du journal dont la valeur Source est égale à `Harness`, entrez **Harness** dans la barre de recherche.
- Pour supprimer tous les caractères d'une zone d'en-tête de colonne, cliquez sur le X de cette zone d'en-tête de colonne. Supprimer tous les caractères d'une zone d'en-tête de colonne revient à entrer * dans cette zone d'en-tête de colonne.

Pour télécharger toutes les informations de journal de l'appareil, y compris toutes les suites et tous les tests que vous avez exécutés, choisissez Télécharger les journaux.

Statuts des résultats des tests Device Farm

La console Device Farm affiche des icônes qui vous aident à évaluer rapidement l'état de votre cycle de test terminé. Pour plus d'informations sur les tests dans Device Farm, consultez [Rapports dans AWS Device Farm](#).

Rubriques

- [Statuts d'un test individuel](#)
- [Statuts pour plusieurs tests](#)

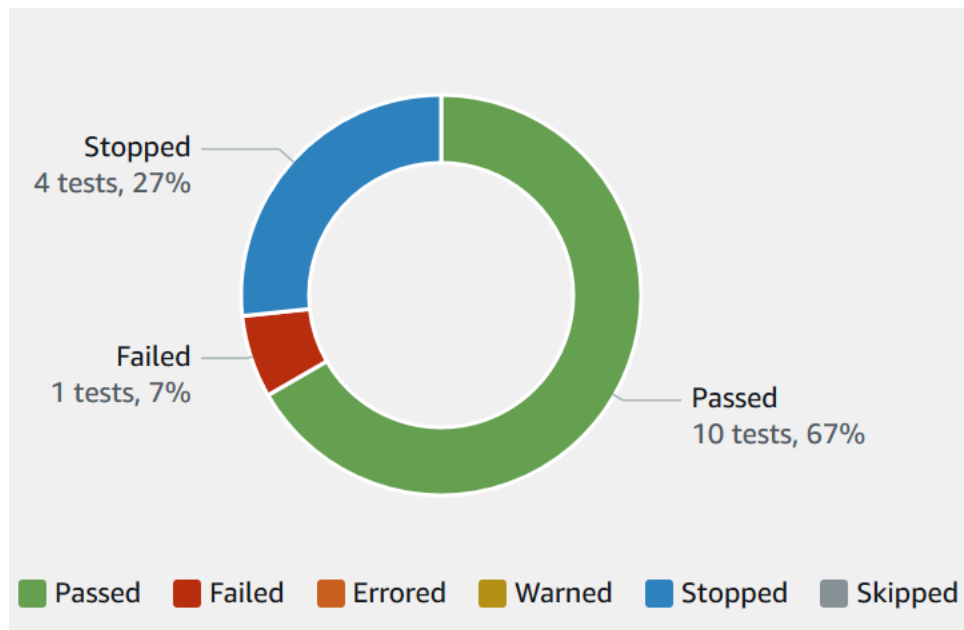
Statuts d'un test individuel

Pour les rapports décrivant un test individuel, Device Farm affiche une icône représentant l'état des résultats du test :

Description	Icône
Le test a abouti.	
Le test a échoué.	
Device Farm a sauté le test.	
Le test s'est arrêté.	
Device Farm a renvoyé un avertissement.	
Device Farm a renvoyé un message d'erreur.	

Statuts pour plusieurs tests

Si vous choisissez une exécution terminée, Device Farm affiche un graphique récapitulatif indiquant le pourcentage de tests dans différents états.



Par exemple, ce graphique des résultats de test montre que 4 tests ont été interrompus, 1 test ayant échoué et 10 tests réussis.

Les graphiques sont toujours codés par couleur et étiquetés.

Téléchargement d'artefacts dans Device Farm

Device Farm rassemble des artefacts tels que des rapports, des fichiers journaux et des images pour chaque test en cours d'exécution.

Vous pouvez télécharger les artefacts créés au cours de votre série de tests :

Fichiers

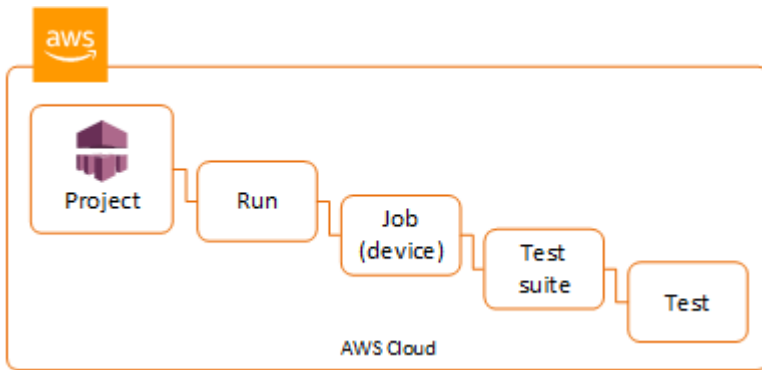
Fichiers générés pendant le test, y compris les rapports Device Farm. Pour de plus amples informations, veuillez consulter [Afficher les rapports de test dans Device Farm](#).

Journaux

Résultat de chaque test de la série de tests.

Captures d'écran

Images d'écran enregistrées pour chaque test de la série de tests.



Télécharger des artefacts (console)

1. Sur la page des rapports de test, à partir de Devices (Appareils), choisissez un appareil mobile.
2. Pour télécharger un fichier, choisissez-en un dans la liste Files (Fichiers).
3. Pour télécharger les journaux de votre série de tests, à partir de Logs (Journaux), choisissez Download logs (Télécharger les journaux).
4. Pour télécharger une capture d'écran, choisissez-en une à partir de Screenshots (Captures d'écran).

Pour plus d'informations sur le téléchargement d'artefacts dans un environnement de test personnalisé, consultez [Téléchargement d'artefacts dans un environnement de test personnalisé](#).

Télécharger des artefacts (AWS CLI)

Vous pouvez utiliser le AWS CLI pour répertorier les artefacts de vos tests.

Rubriques

- [Étape 1 : Obtenez vos noms de ressources Amazon \(ARN\)](#)
- [Étape 2 : Répertoriez vos artefacts](#)
- [Étape 3 : Téléchargez vos artefacts](#)

Étape 1 : Obtenez vos noms de ressources Amazon (ARN)

Vous pouvez répertorier vos artefacts par exécution, tâche, série de tests ou test. Vous avez besoin de l'ARN correspondant. Ce tableau indique l'ARN d'entrée pour chacune des commandes de la AWS CLI liste :

AWS CLI Commande de liste	ARN requis
list-projects	Cette commande renvoie tous les projets et ne requiert aucun ARN.
list-runs	project
list-jobs	run
list-suites	job
list-tests	suite

Par exemple, pour trouver l'ARN d'un test, exécutez list-tests en utilisant l'ARN de votre suite de tests comme paramètre d'entrée.

Exemple :

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

La réponse inclut l'ARN de chaque test de la série de tests.

```
{
  "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
      }
    ]
  }
}

```

Étape 2 : Répertoriez vos artefacts

La commande AWS CLI [list-artefacts](#) renvoie une liste d'artefacts, tels que des fichiers, des captures d'écran et des journaux. Chaque artefact possède une URL vous permettant de télécharger le fichier.

- Appelez la commande `list-artifacts` en spécifiant un ARN d'exécution, de tâche, de série de tests ou de test. Spécifiez le type `FILE`, `LOG` ou `SCREENSHOT`.

Cet exemple renvoie une URL de téléchargement pour chaque artefact disponible pour un test :

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

La réponse contient une URL de téléchargement pour chaque artefact.

```

{
  "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",
    }
  ]
}

```

Étape 3 : Téléchargez vos artefacts

- Téléchargez votre artefact à l'aide de l'URL de l'étape précédente. Cet exemple utilise `curl` pour télécharger un fichier de sortie Appium Java Android :

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

Télécharger des artefacts (API)

La [ListArtifacts](#) méthode Device Farm API renvoie une liste d'artefacts, tels que des fichiers, des captures d'écran et des journaux. Chaque artefact possède une URL vous permettant de télécharger le fichier.

Téléchargement d'artefacts dans un environnement de test personnalisé

Dans un environnement de test personnalisé, Device Farm collecte des artefacts tels que des rapports personnalisés, des fichiers journaux et des images. Ces artefacts sont disponibles pour chaque appareil de la série de tests.

Vous pouvez télécharger les artefacts suivants créés au cours de votre série de tests :

Résultat de spécification de test

Résultat de l'exécution des commandes du fichier YAML de spécification de test.

Artefacts client

Fichier compressé qui contient les artefacts de la série de tests. Il est configuré dans la section `artifacts` de votre fichier YAML de spécification de test.

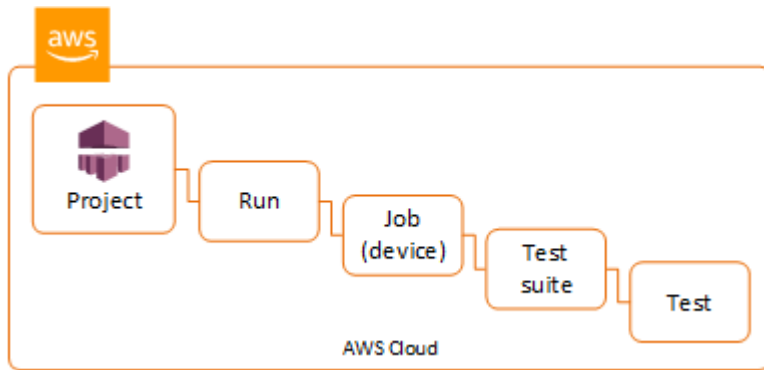
Script shell de spécification de test

Fichier de script shell intermédiaire créé à partir de votre fichier YAML. Étant donné qu'il est utilisé dans la série de tests, le fichier de script shell peut être utilisé pour déboguer le fichier YAML.

Fichier de spécification de test

Fichier YAML utilisé dans la série de tests.

Pour de plus amples informations, veuillez consulter [Téléchargement d'artefacts dans Device Farm](#).



Balisage des ressources AWS Device Farm

AWS Device Farm fonctionne avec l'API AWS Resource Groups Tagging. Cette API vous permet de gérer les ressources de votre compte AWS avec des balises. Vous pouvez ajouter des balises aux ressources, telles que des projets et des tests.

Vous pouvez utiliser des balises pour :

- Organiser votre facture AWS afin de refléter votre propre structure de coût. Pour ce faire, inscrivez-vous pour obtenir votre facture de compte AWS avec les valeurs de clé de balise incluses. Ensuite, pour voir le coût de vos ressources combinées, organisez vos informations de facturation en fonction des ressources possédant les mêmes valeurs de clé de balise. Par exemple, vous pouvez baliser plusieurs ressources avec un nom d'application, puis organiser vos informations de facturation pour afficher le coût total de cette application dans plusieurs services. Pour de plus amples informations, veuillez consulter [Répartition et balisage des coûts](#) dans À propos de la facturation et de la gestion des coûts AWS.
- Contrôler l'accès via des stratégies IAM. Pour ce faire, créez une stratégie qui permet d'accéder à une ressource ou à un ensemble de ressources à en utilisant une condition de valeur de balise.
- Identifier et gérer les exécutions ayant certaines propriétés en tant que balises, telles que la branche utilisée pour les tests.

Pour plus d'informations sur les ressources de balisage, consultez le livre blanc sur [les bonnes pratiques de balisage](#) .

Rubriques

- [Balisage de ressources](#)
- [Recherche de ressources par tag](#)
- [Suppression de balises de ressources](#)

Balisage de ressources

L'API de balisage de groupes de ressources AWS vous permet d'ajouter, de supprimer ou de modifier des balises sur les ressources. Pour plus d'informations, consultez le document [AWS Resource Group Tagging API Reference](#).

Pour baliser une ressource, utilisez l'opération [TagResources](#) à partir du point de terminaison `resourcegroupstaggingapi`. Cette opération prend une liste ARNs des services pris en charge et une liste de paires clé-valeur. La valeur est facultative. Une chaîne vide indique qu'il ne doit pas y avoir de valeur pour cette balise. Par exemple, l'exemple Python suivant balise une série de projets ARNs avec la balise `build-config` contenant la valeur `release` :

```
import boto3

client = boto3.client('resourcegroupstaggingapi')

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

La valeur de balise n'est pas requise. Pour définir une balise sans valeur, utilisez une chaîne vide ("") lorsque vous spécifiez une valeur. Une balise ne peut avoir qu'une seule valeur. Toute valeur de balise antérieure correspondant à une ressource sera remplacée par la nouvelle valeur.

Recherche de ressources par tag

Pour rechercher des ressources par leurs balises, utilisez l'opération `GetResources` à partir du point de terminaison `resourcegroupstaggingapi`. Cette opération prend une série de filtres, dont aucun n'est requis, et renvoie les ressources qui correspondent aux critères donnés. Sans filtres, toutes les ressources balisées sont renvoyées. L'opération `GetResources` vous permet de filtrer les ressources en fonction de

- Valeur de balise
- Type de ressource (par exemple, `devicefarm:run`)

Pour plus d'informations, consultez le document [AWS Resource Group Tagging API Reference](#).

L'exemple suivant recherche les sessions de test du navigateur de bureau Device Farm (`devicefarm:testgrid-sessionresources`) stack dont la balise a la valeur `production` :

```
import boto3
```

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

```

Suppression de balises de ressources

Pour supprimer une balise, utilisez l'opération `UntagResources`, en spécifiant une liste de ressources et les balises à supprimer :

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

```

Frameworks de test et tests intégrés dans AWS Device Farm

Cette section décrit le support de Device Farm pour les frameworks de test et les types de tests intégrés.

Device Farm exécute des tests automatisés en vous demandant de télécharger votre application et vos tests dans un compartiment Amazon S3 sécurisé géré par le service. Une fois chargé, il active l'infrastructure sous-jacente, y compris les [hôtes de test](#) gérés par les services, et exécute les tests en parallèle sur plusieurs appareils. Les résultats des tests sont stockés dans un compartiment S3 géré par le service. Cette architecture, appelée exécution côté service, constitue un moyen rapide et efficace d'exécuter des tests sur des hôtes physiquement proches de l'appareil, sans avoir à gérer vous-même l'infrastructure hôte de test. Cette approche s'adapte bien aux tests indépendants sur de nombreux appareils, ainsi qu'aux tests dans le contexte d'un CI/CD pipeline.

Pour plus d'informations sur la manière dont Device Farm exécute les tests, consultez [Environnements de test dans AWS Device Farm](#).

Note

Pour les testeurs Appium, vous préférerez peut-être exécuter vos tests Appium depuis votre environnement local. Avec une [session d'accès à distance](#), vous pouvez exécuter des tests Appium côté client. Pour plus d'informations, veuillez consulter les tests [Appium côté client](#).

Frameworks de test

Device Farm prend en charge les frameworks de test d'automatisation mobiles suivants :

Cadres de test d'applications Android

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

Cadres de test d'applications iOS

- [Tests Appium automatiques](#)

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

Cadres de test d'applications Web

Les applications web sont prises en charge à l'aide d'Appium. Pour plus d'informations sur l'apport de vos tests à Appium, reportez-vous à la section [Exécuter automatiquement des tests Appium dans Device Farm](#).

Frameworks dans un environnement de test personnalisé

Device Farm ne fournit pas d'assistance pour la personnalisation de l'environnement de test pour le XCTest framework. Pour de plus amples informations, veuillez consulter [Environnements de test personnalisés dans AWS Device Farm](#).

Support des versions d'Appium

Pour les tests exécutés dans un environnement personnalisé, Device Farm prend en charge la version 1 d'Appium. Pour de plus amples informations, veuillez consulter [Environnements de test dans AWS Device Farm](#).

Types de tests intégrés

Grâce aux tests intégrés, vous pouvez tester votre application sur plusieurs appareils sans avoir à écrire et à gérer des scripts d'automatisation des tests. Device Farm propose un type de test intégré :

- [Intégré : Fuzz \(Android et iOS\)](#)

Exécuter automatiquement des tests Appium dans Device Farm

Note

Cette page décrit l'exécution de tests Appium dans l'environnement d'exécution géré côté serveur de Device Farm. [Pour exécuter des tests Appium depuis votre environnement local côté client lors d'une session d'accès à distance, voir Tests Appium côté client.](#)

Cette section décrit comment configurer, emballer et télécharger vos tests Appium pour les exécuter dans l'environnement géré côté serveur de Device Farm. Appium est un outil open source permettant d'automatiser les applications Web natives et mobiles. Pour plus d'informations, consultez la page de [présentation d'Appium](#) sur le site web Appium.

Pour un exemple d'application et des liens vers des tests fonctionnels, voir [Device Farm Sample App pour Android](#) et [Device Farm Sample App pour iOS](#) sur GitHub.

Pour plus d'informations sur les tests dans Device Farm et sur le fonctionnement côté serveur, consultez [Frameworks de test et tests intégrés dans AWS Device Farm](#)

Sélection d'une version d'Appium

Note

Support pour des versions spécifiques d'Appium, des pilotes Appium ou de la programmation SDKs dépend du périphérique et de l'hôte de test sélectionnés pour le test.

Les hôtes de test Device Farm sont préinstallés avec Appium afin de permettre une configuration plus rapide des tests pour des cas d'utilisation plus simples. Cependant, l'utilisation du fichier de spécifications de test vous permet d'installer différentes versions d'Appium si nécessaire.

Scénario 1 : version préconfigurée d'Appium

Device Farm est préconfiguré avec différentes versions du serveur Appium en fonction de l'hôte de test. L'hôte est fourni avec des outils qui activent la version préconfigurée avec le pilote par défaut de la plate-forme de l'appareil (UiAutomator2 pour Android et pour XCUITest iOS).

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

Pour consulter la liste des logiciels pris en charge, consultez la rubrique sur [Logiciels pris en charge dans des environnements de test personnalisés](#).

Scénario 2 : version personnalisée d'Appium

Pour sélectionner une version personnalisée d'Appium, utilisez la npm commande pour l'installer. L'exemple suivant montre comment installer la dernière version d'Appium 2.

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

Scénario 3 : Appium sur les anciens hôtes iOS

Sur le [Ancien hôte de test iOS](#), vous pouvez choisir des versions spécifiques d'Appium avec. avm Par exemple, pour utiliser la avm commande permettant de définir la version du serveur Appium sur 2.1.2, ajoutez ces commandes à votre fichier YAML de spécifications de test.

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

Sélection d'une WebDriverAgent version pour les tests iOS

Pour exécuter des tests Appium sur des appareils iOS, l'utilisation de WebDriverAgent est requise. Cette application doit être signée pour être installée sur les appareils iOS. Device Farm fournit des versions pré-signées WebDriverAgent qui sont disponibles lors des exécutions d'environnements de test personnalisés.

L'extrait de code suivant peut être utilisé pour sélectionner une WebDriverAgent version de Device Farm dans votre fichier de spécifications de test compatible avec la version de votre pilote d'XCTestinterface utilisateur.

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

```

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

```

[Pour plus d'informations à ce sujet WebDriverAgent, consultez la documentation d'Appium.](#)

Intégrer les tests Appium à Device Farm

Suivez les instructions suivantes pour intégrer les tests Appium à AWS Device Farm. Pour plus d'informations sur l'utilisation des tests Appium dans Device Farm, consultez. [Exécuter automatiquement des tests Appium dans Device Farm](#)

Configurez votre package de test Appium

Utilisez les instructions suivantes pour configurer votre package de test.

Java (JUnit)

1. Modifiez `pom.xml` pour définir l'empaquetage dans un fichier JAR :

```

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

```

2. Modifiez `pom.xml` pour utiliser `maven-jar-plugin` afin de créer vos tests dans un fichier JAR.

Le plugin suivant crée votre code source de test (tout ce qui se trouve dans le `src/test` répertoire) dans un fichier JAR :

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

3. Modifiez `pom.xml` pour utiliser `maven-dependency-plugin` afin de créer des dépendances en tant que fichiers JAR.

Le plugin suivant copie vos dépendances dans le `dependency-jars` répertoire :

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

4. Enregistrez l'assemblage XML suivant dans `src/main/assembly/zip.xml`.

Le code XML suivant est une définition d'assemblage qui, une fois configurée, indique à Maven de créer un fichier .zip contenant tout ce qui se trouve à la racine du répertoire de sortie de votre build et du répertoire : `dependency-jars`

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

5. Modifiez le fichier `pom.xml` pour utiliser `maven-assembly-plugin` afin de compresser tous les tests et les dépendances en un seul fichier .zip.

Le plug-in suivant utilise l'assemblage précédant pour créer un fichier .zip nommé `zip-with-dependencies` dans le répertoire de sortie de build à chaque fois que la commande `mvn package` est exécutée :

```
<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

Si un message d'erreur indique que l'annotation n'est pas prise en charge dans la version 1.3, ajoutez les éléments suivants au fichier `pom.xml` :

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

Java (TestNG)

1. Modifiez `pom.xml` pour définir l'empaquetage dans un fichier JAR :

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

2. Modifiez `pom.xml` pour utiliser `maven-jar-plugin` afin de créer vos tests dans un fichier JAR.

Le plugin suivant crée votre code source de test (tout ce qui se trouve dans le `src/test` répertoire) dans un fichier JAR :

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

3. Modifiez `pom.xml` pour utiliser `maven-dependency-plugin` afin de créer des dépendances en tant que fichiers JAR.

Le plugin suivant copie vos dépendances dans le `dependency-jars` répertoire :

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

4. Enregistrez l'assemblage XML suivant dans `src/main/assembly/zip.xml`.

Le code XML suivant est une définition d'assemblage qui, une fois configurée, indique à Maven de créer un fichier `.zip` contenant tout ce qui se trouve à la racine du répertoire de sortie de votre build et du répertoire : `dependency-jars`

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

5. Modifiez le fichier `pom.xml` pour utiliser `maven-assembly-plugin` afin de compresser tous les tests et les dépendances en un seul fichier `.zip`.

Le plug-in suivant utilise l'assemblage précédant pour créer un fichier `.zip` nommé `zip-with-dependencies` dans le répertoire de sortie de build à chaque fois que la commande `mvn package` est exécutée :

```
<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

Si un message d'erreur indique que l'annotation n'est pas prise en charge dans la version 1.3, ajoutez les éléments suivants au fichier `pom.xml` :

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

Node.JS

Pour emballer vos tests Appium Node.js et les télécharger sur Device Farm, vous devez installer les éléments suivants sur votre machine locale :

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

Utilisez cet outil pour développer et créer vos packages de test afin qu'aucune dépendance inutile n'y soit incluse.

- Node.js
- npm-bundle (installé globalement)

1. Vérifiez que nvm est présent

```
command -v nvm
```

Vous devez voir nvm en tant que sortie.

Pour plus d'informations, consultez [nvm](#) on GitHub.

2. Pour installer Node.js, exécutez la commande suivante :

```
nvm install node
```

Vous pouvez spécifier une version particulière de Node.js :

```
nvm install 11.4.0
```

3. Vérifiez que la version correcte de Node est en cours d'utilisation :

```
node -v
```

4. Installer npm-bundle globalement :

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

Python

1. Nous vous recommandons vivement de configurer [virtualenv Python](#) pour le développement et la création d'un package de tests afin qu'aucune dépendance inutile ne soit incluse dans votre package d'application.

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

```
$ source bin/activate
```

 Tip

- Ne créez pas d'environnement virtualenv Python avec l'option `--system-site-packages`, car il hérite des packages de votre répertoire `site-packages` global. Cela peut entraîner l'inclusion dans votre environnement virtuel de dépendances qui ne sont pas requises par vos tests.
- Vous devez également vérifier que vos tests n'utilisent pas de dépendances aux bibliothèques natives. En effet, il est possible que ces bibliothèques natives ne soient pas présentes sur l'instance sur laquelle ces tests sont exécutés.

2. Installez `py.test` dans votre environnement virtuel.

```
$ pip install pytest
```

3. Installez le client Appium Python dans votre environnement virtuel.

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

4. À moins que vous ne spécifiiez un chemin différent en mode personnalisé, Device Farm s'attend à ce que vos tests y soient stockés `tests/`. Vous pouvez utiliser `find` pour afficher tous les fichiers dans un dossier :

```
$ find tests/
```

Vérifiez que ces fichiers contiennent des suites de test que vous souhaitez exécuter sur Device Farm

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

5. Exécutez cette commande à partir du dossier `workspace` de votre environnement virtuel pour afficher une liste de vos tests sans les exécuter.

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


Vérifiez que le résultat indique les tests que vous souhaitez exécuter sur Device Farm.

6. Nettoyez tous les fichiers mis en cache sous votre dossier tests/ :

```
$ 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. Exécutez la commande suivante dans votre espace de travail pour générer le fichier requirements.txt :

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

Ruby

Pour empaqueter vos tests Appium Ruby et les télécharger sur Device Farm, vous devez installer les éléments suivants sur votre machine locale :

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

Utilisez cet outil de ligne de commande pour développer et créer vos packages de test afin qu'aucune dépendance inutile n'y soit incluse.

- Ruby
- Bundler (Cette gem est généralement installée avec Ruby.)

1. Installez les clés nécessaires, RVM et Ruby. Pour plus d'informations, consultez la [section relative à l'installation de RVM](#) sur le site Web RVM.

Une fois l'installation terminée, rechargez votre terminal. Pour cela, déconnectez-vous puis reconnectez-vous.



Note

RVM est chargé en tant que fonction pour le shell bash uniquement.

2. Vérifiez que rvm est installé correctement.

```
command -v rvm
```

Vous devez voir `rvm` en tant que sortie.

3. Si vous souhaitez installer une version spécifique de Ruby, par exemple **2.5.3**, exécutez la commande suivante :

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

Vérifiez que vous êtes sur la version demandée de Ruby :

```
ruby -v
```

4. Configurez le bundler pour compiler les packages pour les plateformes de test souhaitées :

```
bundle config specific_platform true
```

5. Mettez à jour votre fichier `.lock` pour ajouter les plateformes nécessaires à l'exécution des tests.

- Si vous compilez des tests à exécuter sur des appareils Android, exécutez cette commande pour configurer le Gemfile afin qu'il utilise les dépendances de l'hôte de test Android :

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

- Si vous compilez des tests à exécuter sur des appareils iOS, exécutez cette commande pour configurer le Gemfile afin qu'il utilise les dépendances de l'hôte de test iOS :

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

6. La gem bundler est généralement installée par défaut. Si ce n'est pas le cas, installez-la :

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

Création d'un fichier de package de test compressé

Warning

Dans Device Farm, la structure des dossiers des fichiers de votre package de test compressé est importante, et certains outils d'archivage modifieront implicitement la structure de votre fichier ZIP. Nous vous recommandons de suivre les utilitaires de ligne de commande spécifiés ci-dessous plutôt que d'utiliser les utilitaires d'archivage intégrés au gestionnaire de fichiers de votre bureau local (tels que le Finder ou l'Explorateur Windows).

Maintenant, groupez vos tests pour la batterie de périphériques.

Java (JUnit)

Construisez et empaquetez vos tests :

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

Le fichier `zip-with-dependencies.zip` sera créé en conséquence. Ceci est votre package de tests.

Java (TestNG)

Construisez et empaquetez vos tests :

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

Le fichier `zip-with-dependencies.zip` sera créé en conséquence. Ceci est votre package de tests.

Node.JS

1. Vérifiez votre projet.

Assurez-vous d'être dans le répertoire racine de votre projet. Vous pouvez voir `package.json` dans le répertoire racine.

2. Pour installer vos dépendances locales, exécutez la commande suivante.

```
npm install
```

Cette commande crée également un dossier `node_modules` au sein de votre répertoire actuel.

 Note

À ce stade, vous devriez être en mesure d'exécuter vos tests en local.

3. Exécutez cette commande pour regrouper les fichiers de votre dossier actif dans un package `*.tgz`. Le fichier créé est nommé en fonction de la propriété `name` indiquée dans votre fichier `package.json`.

```
npm-bundle
```

Ce fichier tarball (`.tgz`) contient votre code et toutes les dépendances.

4. Exécutez cette commande pour inclure le fichier tarball (`*.tgz`) généré à l'étape précédente dans une seule archive compressée :

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

Il s'agit du `MyTests.zip` fichier que vous chargez sur Device Farm dans le cadre de la procédure suivante.

Python

Python 2

Générez une archive des packages Python requis (appelé dossier `wheelhouse`) en utilisant `pip` :

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

Regroupez votre dossier `wheelhouse`, vos tests et vos exigences concernant `pip` dans une archive `zip` pour Device Farm :

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

Python 3

Regroupez vos tests et vos exigences concernant pip dans un fichier zip :

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

Ruby

1. Pour créer un environnement Ruby virtuel, exécutez cette commande :

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

2. Pour utiliser l'environnement que vous venez de créer, exécutez cette commande :

```
rvm gemset use myGemset
```

3. Vérifiez votre code source.

Assurez-vous d'être dans le répertoire racine de votre projet. Vous pouvez voir Gemfile dans le répertoire racine.

4. Pour installer vos dépendances locales et toutes les gems à partir du fichier Gemfile, exécutez cette commande :

```
bundle install
```

Note

À ce stade, vous devriez être en mesure d'exécuter vos tests en local. Pour exécuter un test en local, utilisez cette commande :

```
bundle exec $test_command
```

5. Regroupez vos gems dans le dossier vendor/cache.

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

6. Exécutez la commande suivante pour regrouper votre code source, ainsi que toutes vos dépendances, dans une seule archive compressée :

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

Il s'agit du `MyTests.zip` fichier que vous chargez sur Device Farm dans le cadre de la procédure suivante.

Téléchargez votre package de test sur Device Farm

Vous pouvez utiliser la console Device Farm pour télécharger vos tests.

1. Connectez-vous à la console Device Farm à l'adresse <https://console.aws.amazon.com/devicefarm>.
2. Sur le panneau de navigation de Device Farm, choisissez Mobile Device Testing, puis Projects.
3. Si vous êtes un nouvel utilisateur, choisissez Nouveau projet, entrez un nom pour le projet, puis choisissez Soumettre.

Si vous avez déjà un projet, vous pouvez le choisir pour y télécharger vos tests.

4. Ouvrez votre projet, puis choisissez Create run.
5. Sous Paramètres d'exécution, attribuez un nom approprié à votre test. Il peut contenir n'importe quelle combinaison d'espaces ou de ponctuation.
6. Pour les tests natifs Android et iOS

Sous Paramètres d'exécution, choisissez application Android si vous testez une application Android (.apk) ou application iOS si vous testez une application iOS (.ipa). Ensuite, sous Sélectionner une application, sélectionnez Télécharger votre propre application pour télécharger le package distribuable de votre application.

Note

Le fichier doit être un fichier Android .apk ou un fichier iOS .ipa. Les applications iOS doivent être construites pour des appareils réels, et non pour le simulateur.

Pour les tests d'application web mobile

Sous Paramètres d'exécution, sélectionnez Web App.

7. Sous Configurer le test, dans la section Sélectionner le framework de test, choisissez le framework Appium avec lequel vous testez, puis téléchargez votre propre package de test.
8. Recherchez et choisissez le fichier .zip contenant vos tests. Le fichier .zip doit respecter le format décrit dans [Configurez votre package de test Appium](#).
9. Suivez les instructions pour sélectionner les appareils et démarrer l'exécution. Pour de plus amples informations, veuillez consulter [Création d'un test dans Device Farm](#).

Note

Device Farm ne modifie pas les tests Appium.

Prenez des captures d'écran de vos tests (facultatif)

Vous pouvez effectuer des captures d'écran dans le cadre de vos tests.

Device Farm définit la propriété `DEVICEFARM_SCREENSHOT_PATH` sur un chemin d'accès complet du système de fichiers local, sur lequel Device Farm prévoit que les captures d'écran Appium seront enregistrées. Le répertoire spécifique au test dans lequel les captures d'écran sont stockées est défini lors de l'exécution. Les captures d'écran sont extraites automatiquement dans vos rapports Device Farm. Pour afficher les captures d'écran, dans la console Device Farm, choisissez la section Screenshots (Captures d'écran).

Pour plus d'informations sur la prise de captures d'écran dans les tests Appium, voir [Take Screenshot](#) dans la documentation Appium API.

Tests Android dans AWS Device Farm

Device Farm prend en charge plusieurs types de tests d'automatisation pour les appareils Android, ainsi que deux tests intégrés.

Pour plus d'informations sur les tests dans Device Farm, consultez [Frameworks de test et tests intégrés dans AWS Device Farm](#).

Cadres de test d'applications Android

Les tests suivants sont disponibles pour les appareils Android.

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

Types de tests intégrés pour Android

Il existe un type de test intégré disponible pour les appareils Android :

- [Intégré : Fuzz \(Android et iOS\)](#)

Instrumentation pour Android et AWS Device Farm

Device Farm prend en charge l'instrumentation (EspressoJUnit, Robotium ou tout autre test basé sur l'instrumentation) pour Android.

Device Farm fournit également un exemple d'application Android et des liens vers des tests fonctionnels dans trois frameworks d'automatisation Android, dont Instrumentation (Espresso). L'[exemple d'application Device Farm pour Android](#) est disponible en téléchargement sur GitHub.

Pour plus d'informations sur les tests dans Device Farm, consultez [Frameworks de test et tests intégrés dans AWS Device Farm](#).

Rubriques

- [Qu'est-ce que l'instrumentation ?](#)
- [Considérations relatives aux tests d'instrumentation Android](#)
- [Analyse syntaxique des tests en mode standard](#)
- [Intégrer l'instrumentation Android à Device Farm](#)

Qu'est-ce que l'instrumentation ?

Instrumentation Android vous permet d'appeler des méthodes de rappel dans votre code de test, afin que vous puissiez parcourir le cycle de vie d'un composant étape par étape, comme si vous déboguez le composant. Pour plus d'informations, consultez la section [Tests instrumentés](#) dans la section Types et emplacements de tests de la documentation des outils de développement Android.

Considérations relatives aux tests d'instrumentation Android

Lorsque vous utilisez l'instrumentation Android, tenez compte des recommandations et remarques suivantes.

Vérifiez la compatibilité du système d'exploitation Android

Consultez la [documentation Android](#) pour vous assurer que l'instrumentation est compatible avec la version de votre système d'exploitation Android.

Exécution depuis la ligne de commande

Pour exécuter des tests d'instrumentation depuis la ligne de commande, veuillez suivre la [documentation Android](#).

System Animations (Animations système)

Selon la [documentation Android pour les tests d'Espresso](#), il est recommandé de désactiver les animations du système lors des tests sur des appareils réels. Device Farm désactive automatiquement les paramètres Window Animation Scale, Transition Animation Scale et Animator Duration Scale lorsqu'il s'exécute avec le testeur d'instrumentation [JUnitandroid.support.test.runner.Android](#) Runner.

Test Recorders (Enregistreurs de test)

Device Farm prend en charge les frameworks, tels que Robotium, dotés d'outils record-and-playback de script.

Analyse syntaxique des tests en mode standard

En mode standard d'exécution, Device Farm analyse votre suite de tests et identifie les classes et méthodes de test uniques qu'elle exécutera. Cela se fait par le biais d'un outil appelé [Dex Test Parser](#).

Lorsqu'un fichier .apk d'instrumentation Android est fourni en entrée, l'analyseur renvoie les noms de méthode complets des tests qui correspondent aux conventions JUnit 3 et JUnit 4.

Pour tester cela dans un environnement local :

1. Téléchargez le [dex-test-parser](#) binaire.
2. Exécutez la commande suivante pour obtenir la liste des méthodes de test qui seront exécutées sur Device Farm :

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

Intégrer l'instrumentation Android à Device Farm

Note

Suivez les instructions suivantes pour intégrer les tests d'instrumentation Android à AWS Device Farm. Pour plus d'informations sur l'utilisation des tests d'instrumentation dans Device Farm, consultez [Instrumentation pour Android et AWS Device Farm](#).

Téléchargez vos tests d'instrumentation Android

Utilisez la console Device Farm pour télécharger vos tests.

1. Connectez-vous à la console Device Farm à l'adresse <https://console.aws.amazon.com/devicefarm>.
2. Sur le panneau de navigation de Device Farm, choisissez Mobile Device Testing, puis Projects.
3. Dans la liste des projets, choisissez le projet dans lequel vous souhaitez télécharger vos tests.

Tip

Vous pouvez utiliser la barre de recherche pour filtrer la liste des projets par nom. Pour créer un projet, suivez les instructions fournies dans [Création d'un projet dans AWS Device Farm](#).

4. Sélectionnez Créer une course.
5. Sous Sélectionner une application, dans la section Options de sélection des applications, sélectionnez Télécharger votre propre application.
6. Recherchez et sélectionnez votre fichier d'application Android. Le fichier doit être au format .apk.
7. Sous Configurer le test, dans la section Sélectionner le cadre de test, choisissez Instrumentation, puis sélectionnez Choisir un fichier.
8. Recherchez et choisissez le fichier .apk contenant vos tests.
9. Suivez les instructions restantes pour sélectionner les appareils et démarrer l'exécution.

(Facultatif) Prenez des captures d'écran lors des tests d'instrumentation Android

Vous pouvez effectuer des captures d'écran dans le cadre de vos tests Instrumentation Android.

Pour effectuer des captures d'écran, appelez l'une des méthodes suivantes :

- Pour Robotium, appelez la méthode `takeScreenShot` (par exemple, `solo.takeScreenShot();`).
- Pour Spoon, appelez la méthode `screenshot`, par exemple :

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

Lors d'un test, Device Farm obtient des captures d'écran des emplacements suivants sur les appareils, s'ils existent, puis les ajoute aux rapports de test :

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

Tests iOS dans AWS Device Farm

Device Farm prend en charge plusieurs types de tests d'automatisation pour les appareils iOS, ainsi qu'un test intégré.

Pour plus d'informations sur les tests dans Device Farm, consultez [Frameworks de test et tests intégrés dans AWS Device Farm](#).

Cadres de test d'applications iOS

Les tests suivants sont disponibles pour les appareils iOS.

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

Types de tests intégrés pour iOS

Actuellement, il n'existe qu'un type de test intégré disponible pour les appareils iOS.

- [Intégré : Fuzz \(Android et iOS\)](#)

Intégration de Device Farm à XCTest pour iOS

Avec Device Farm, vous pouvez utiliser le XCTest framework pour tester votre application sur de vrais appareils. Pour plus d'informations XCTest, consultez la section [Principes de base des tests](#) dans Tester avec Xcode.

Pour exécuter un test, vous créez les packages pour votre test et vous les téléchargez sur Device Farm.

Pour plus d'informations sur les tests dans Device Farm, consultez [Frameworks de test et tests intégrés dans AWS Device Farm](#).

Rubriques

- [Créez les packages pour votre XCTest course](#)
- [Téléchargez les packages pour votre XCTest course sur Device Farm](#)

Créez les packages pour votre XCTest course

Pour tester votre application à l'aide du XCTest framework, Device Farm a besoin des éléments suivants :

- Votre package d'application en tant que fichier .ipa.
- Votre XCTest package sous forme de .zip fichier.

Vous créez ces packages en utilisant la sortie de build générée par Xcode. Procédez comme suit pour créer les packages afin de pouvoir les télécharger sur Device Farm.

Pour générer la sortie de build pour votre application.

1. Ouvrez votre projet d'application dans Xcode.
2. Dans le menu déroulant de méthode de la barre d'outils Xcode, choisissez Generic iOS Device (Appareil iOS générique) comme destination.

3. Dans le menu Product (Produit), choisissez Build For (Build pour), puis Testing (Test).

Pour créer le package d'application

1. Dans le navigateur de projet dans Xcode, sous Products (Produits), ouvrez le menu contextuel pour le fichier nommé *app-project-name*.app. Choisissez ensuite Show in Finder (Afficher dans l'outil de recherche). L'outil de recherche s'ouvre dans un dossier nommé Debug-iphonios, qui contient la sortie générée par Xcode pour votre build de test. Ce dossier inclut votre fichier .app.
2. Dans l'outil de recherche, créez un nouveau dossier et nommez-le Payload.
3. Copiez le fichier *app-project-name*.app et collez-le dans le dossier Payload.
4. Ouvrez le menu contextuel pour le dossier Payload et choisissez Compress "Payload" (Compresser « Payload »). Un fichier nommé Payload.zip est créé.
5. Remplacez le nom de fichier et l'extension de Payload.zip par *app-project-name*.ipa.

Dans une étape ultérieure, vous fournirez ce fichier à Device Farm. Pour rendre le fichier plus facile à trouver, vous pouvez le déplacer vers un autre emplacement, tel que votre bureau.

6. Le cas échéant, vous pouvez supprimer le dossier Payload et le fichier .app dans celui-ci.

Pour créer le XCTest package

1. Dans l'outil de recherche, dans le répertoire Debug-iphonios, ouvrez le menu contextuel pour le fichier *app-project-name*.app. Ensuite, choisissez Show Package Contents (Afficher le contenu du package).
2. Dans le contenu du package, ouvrez le dossier Plugins. Ce dossier contient un fichier nommé *app-project-name*.xctest.
3. Ouvrez le menu contextuel pour ce fichier et choisissez Compress (Compresser) "*app-project-name.xctest*". Un fichier nommé *app-project-name*.xctest.zip est créé.

Dans une étape ultérieure, vous fournirez ce fichier à Device Farm. Pour rendre le fichier plus facile à trouver, vous pouvez le déplacer vers un autre emplacement, tel que votre bureau.

Téléchargez les packages pour votre XCTest course sur Device Farm

Utilisez la console Device Farm pour télécharger les packages de votre test.

1. Connectez-vous à la console Device Farm à l'adresse <https://console.aws.amazon.com/devicefarm>.
2. Si vous n'en possédez pas déjà un, créez un projet. Pour connaître les étapes de création d'un projet, consultez [Création d'un projet dans AWS Device Farm](#).

Sinon, dans le panneau de navigation de Device Farm, choisissez Mobile Device Testing, puis Projects.

3. Choisissez le projet que vous souhaitez utiliser pour exécuter le test.
4. Choisissez Create run.
5. Sous Paramètres d'exécution, dans la section Type d'exécution, choisissez application iOS.
6. Sous Sélectionner une application, dans la section Options de sélection des applications, sélectionnez Télécharger votre propre application. Sélectionnez ensuite Choisir un fichier sous Télécharger l'application.
7. Accédez au fichier .ipa pour votre application et chargez-le.

 Note

Votre package .ipa doit être conçu pour les tests.

8. Sous Configurer le test, dans la section Sélectionner le framework de test, sélectionnez XCTest. Sélectionnez ensuite Choisir un fichier sous Télécharger l'application.
9. Accédez au .zip fichier contenant le XCTest package de votre application et téléchargez-le.
10. Effectuez les étapes restantes du processus de création de projet. Vous sélectionnez les appareils sur lesquels vous souhaitez exécuter le test et vous spécifiez leur état.
11. Choisissez Create run. Device Farm exécute votre test et affiche les résultats dans la console.

Intégration de l' XCTest interface utilisateur pour iOS à Device Farm

Device Farm fournit un support pour le framework de test de l' XCTest interface utilisateur. [Plus précisément, Device Farm prend en charge les tests d' XCTest interface utilisateur écrits à la fois en Objective-C et en Swift.](#)

Le framework d' XCTest interface utilisateur permet de tester l'interface utilisateur dans le développement d'iOS, en s'appuyant sur XCTest. Pour plus d'informations, consultez [User Interface Testing](#) dans la bibliothèque pour les développeurs iOS.

Pour obtenir des informations générales sur les tests dans Device Farm, consultez [Frameworks de test et tests intégrés dans AWS Device Farm](#).

Suivez les instructions ci-dessous pour intégrer Device Farm au framework de test de l' XCTest interface utilisateur pour iOS.

Rubriques

- [Préparez vos tests d' XCTest interface utilisateur iOS](#)
- [Option 1 : création d'un package d' XCTest interface utilisateur .ipa](#)
- [Option 2 : création d'un package .zip d' XCTest interface utilisateur](#)
- [Téléchargez vos tests d' XCTest interface utilisateur iOS](#)

Préparez vos tests d' XCTest interface utilisateur iOS

Vous pouvez télécharger un .ipa fichier ou un .zip fichier pour votre package de test XCTEST_UI.

Un .ipa fichier est une archive d'application contenant l'application iOS Runner sous forme de bundle. Des fichiers supplémentaires ne peuvent pas être inclus dans le .ipa fichier.

Si vous importez un .zip fichier, il peut contenir directement l'application iOS Runner ou un .ipa fichier. Vous pouvez également inclure d'autres fichiers dans le .zip fichier si vous souhaitez les utiliser pendant les tests. Par exemple, vous pouvez inclure des fichiers tels que .xctestrun, .xcworkspace ou .xcodeproj dans un .zip fichier, pour exécuter des plans de test XCUI sur une ferme de périphériques. Des instructions détaillées sur la façon d'exécuter les plans de test sont disponibles dans le fichier de spécifications de test par défaut pour le type de test XCUI.

Option 1 : création d'un package d' XCTest interface utilisateur .ipa

Le bundle yourAppNameUITest-Runner.app est produit par Xcode lorsque vous créez votre projet à des fins de test. Vous pouvez la trouver dans le répertoire Products de votre projet.

Pour créer un fichier .ipa :

1. Créez un répertoire appelé *Payload*.
2. Ajoutez le répertoire de votre application au répertoire Payload.
3. Archivez le répertoire Payload dans un .zip fichier, puis remplacez l'extension du fichier par .ipa

La structure de dossiers suivante montre comment un exemple d'application nommé *my-project-nameUITest-Runner.app* serait empaqueté sous forme de .ipa fichier :

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

Option 2 : création d'un package .zip d' XCTest interface utilisateur

Device Farm génère automatiquement un .xctestrun fichier pour exécuter votre suite complète de tests d' XCTest interface utilisateur. Si vous souhaitez utiliser votre propre .xctestrun fichier sur Device Farm, vous pouvez compresser vos .xctestrun fichiers et le répertoire de l'application dans un .zip fichier. Si vous avez déjà un .ipa fichier pour votre package de test, vous pouvez l'inclure ici au lieu de **-Runner.app*.

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

Si vous souhaitez exécuter un plan de test Xcode pour vos tests XCUI sur Device Farm, vous pouvez créer un fichier zip contenant votre fichier my-project-nameUITest-Runner.app ou my-project-nameUITest.ipa et les fichiers de code source xcode nécessaires pour exécuter XCTest_UI avec les plans de test, y compris un fichier ou. .xcworkspace .xcodeproj

Voici un exemple de zip utilisant un .xcodeproj fichier :

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


Voici un exemple de zip utilisant un .xcworkspace fichier :

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

Note

Vérifiez que le package .zip de votre XCTest interface utilisateur ne contient pas de répertoire nommé « Payload ».

Téléchargez vos tests d' XCTest interface utilisateur iOS

Utilisez la console Device Farm pour télécharger vos tests.

1. Connectez-vous à la console Device Farm à l'adresse <https://console.aws.amazon.com/devicefarm>.
2. Sur le panneau de navigation de Device Farm, choisissez Mobile Device Testing, puis Projects.
3. Dans la liste des projets, choisissez le projet dans lequel vous souhaitez télécharger vos tests.

Tip

Vous pouvez utiliser la barre de recherche pour filtrer la liste des projets par nom.
Pour créer un projet, suivez les instructions de [Création d'un projet dans AWS Device Farm](#)

4. Choisissez Create run.

5. Sous Paramètres d'exécution, dans la section Type d'exécution, choisissez application iOS.
6. Sous Sélectionner une application, dans la section Options de sélection des applications, sélectionnez Télécharger votre propre application. Sélectionnez ensuite Choisir un fichier sous Télécharger l'application.
7. Recherchez et sélectionnez votre fichier d'application iOS. Le fichier doit être au format .ipa.

 Note

Assurez-vous que votre fichier .ipa est conçu pour un appareil iOS et non pour un simulateur.

8. Sous Configurer le test, dans la section Sélectionner le framework de test, choisissez XCTest UI. Sélectionnez ensuite Choisir un fichier sous Télécharger l'application.
9. Accédez au fichier .ipa ou .zip qui contient le lanceur de test de l'XCTest interface utilisateur iOS et choisissez-le.
10. Effectuez les étapes restantes du processus de création d'une course. Vous allez sélectionner les appareils sur lesquels vous souhaitez effectuer le test et éventuellement spécifier une configuration supplémentaire.
11. Choisissez Create run. Device Farm exécute votre test et affiche les résultats dans la console.

Tests d'applications Web dans AWS Device Farm

Device Farm propose des tests avec Appium pour les applications Web. Pour plus d'informations sur la configuration de vos tests Appium sur Device Farm, consultez [the section called “Tests Appium automatiques”](#)

Pour plus d'informations sur les tests dans Device Farm, consultez [Frameworks de test et tests intégrés dans AWS Device Farm](#).

Règles relatives aux appareils mesurés et non mesurés

La notion d'appareils limités ou illimités fait référence à la facturation de leur utilisation. Par défaut, les appareils Device Farm sont équipés de compteurs et vous êtes facturé à la minute une fois les minutes d'essai gratuites épuisées. Vous pouvez également choisir d'acheter des appareils illimités, ce qui vous permet d'effectuer des tests de manière illimitée pour un coût mensuel fixe. Pour plus d'informations sur les tarifs, consultez la section [Tarifs d'AWS Device Farm](#).

Si vous choisissez de démarrer une exécution avec un groupe d'appareils contenant à la fois des appareils iOS et des appareils Android, certaines règles s'appliquent selon qu'ils sont limités ou illimités. Par exemple, si vous disposez de cinq appareils Android illimités et de cinq appareils iOS illimités, vos exécutions de test Web utilisent vos appareils illimités.

Voici un autre exemple : supposons que vous disposiez de cinq appareils Android illimités et de 0 appareil iOS illimité. Si vous sélectionnez uniquement les appareils Android pour exécuter votre test Web, vos appareils illimités sont utilisés. Si vous sélectionnez à la fois les appareils Android et iOS pour exécuter votre test Web, le mode de facturation est limité et vos appareils illimités ne sont pas utilisés.

Tests intégrés dans AWS Device Farm

Device Farm prend en charge les types de tests intégrés pour les appareils Android et iOS.

Grâce aux tests intégrés, vous pouvez tester votre application sur plusieurs appareils sans avoir à écrire et à gérer des scripts d'automatisation des tests. Cela peut vous faire gagner du temps et économiser des efforts, en particulier lorsque vous débutez avec Device Farm. Device Farm propose le type de test intégré suivant :

- [Intégré : Fuzz \(Android et iOS\)](#)— Le fuzz test envoie aléatoirement des événements d'interface utilisateur aux appareils, puis rapporte les résultats.

Pour plus d'informations sur les tests et les frameworks de test dans Device Farm, consultez [Frameworks de test et tests intégrés dans AWS Device Farm](#).

Exécution du test de fuzz intégré à Device Farm (Android et iOS)

Le test de fuzz intégré à Device Farm envoie de manière aléatoire les événements de l'interface utilisateur aux appareils, puis communique les résultats.

Pour plus d'informations sur les tests dans Device Farm, consultez [Frameworks de test et tests intégrés dans AWS Device Farm](#).

Pour exécuter le test de fuzz intégré

1. Connectez-vous à la console Device Farm à l'adresse <https://console.aws.amazon.com/devicefarm>.

2. Sur le panneau de navigation de Device Farm, choisissez Mobile Device Testing, puis Projects.
3. Dans la liste des projets, choisissez le projet dans lequel vous souhaitez exécuter le test de fuzz intégré.

 Tip

Vous pouvez utiliser la barre de recherche pour filtrer la liste des projets par nom.

Pour créer un projet, suivez les instructions fournies dans [Création d'un projet dans AWS Device Farm](#).

4. Choisissez Create run.
5. Sous Paramètres d'exécution, sélectionnez votre type d'exécution dans la section Type d'exécution. Sélectionnez une application Android si aucune application n'est prête à être testée ou si vous testez une application Android (.apk). Sélectionnez une application iOS si vous testez une application iOS (.ipa).
6. Sous Select app, choisissez Select sample app fourni par Device Farm si aucune application n'est disponible pour les tests. Si vous apportez votre propre application, sélectionnez Télécharger votre propre application, puis choisissez votre fichier de candidature.
7. Sous Configurer le test, dans la section Sélectionner le framework de test, choisissez Built-in : Fuzz.
8. Si l'un des paramètres suivants s'affiche, vous pouvez accepter les valeurs par défaut ou spécifier vos propres valeurs :
 - Event count (Nombre d'événements) : spécifiez un nombre compris entre 1 et 10 000, qui représente le nombre d'événements d'interface utilisateur que le test Fuzz doit effectuer.
 - Limitation des événements : spécifiez un nombre compris entre 0 et 1 000, représentant le nombre de millisecondes pendant lequel le test de fuzz doit attendre avant de réaliser le prochain événement d'interface utilisateur.
 - Randomizer Seed (Valeur initiale de générateur aléatoire) : spécifiez un nombre que le test Fuzz doit utiliser pour randomiser les événements d'interface utilisateur. Si vous spécifiez le même nombre pour les tests Fuzz suivants, les séquences d'événements seront identiques.
9. Suivez les instructions restantes pour sélectionner les appareils et démarrer l'exécution.

Environnements de test personnalisés dans AWS Device Farm

AWS Device Farm permet de configurer un environnement personnalisé pour les tests automatisés (mode personnalisé), ce qui est l'approche recommandée pour tous les utilisateurs de Device Farm. Pour en savoir plus sur les environnements de Device Farm, consultez la section [Environnements de test](#).

Les avantages du mode personnalisé par rapport au mode standard incluent :

- Exécution plus rapide des end-to-end tests : le package de test n'est pas analysé pour détecter tous les tests de la suite, ce qui permet d'éviter des preprocessing/postprocessing frais supplémentaires.
- Journal en direct et diffusion vidéo : vos journaux de test et vos vidéos côté client sont diffusés en direct lorsque vous utilisez le mode personnalisé. Cette fonctionnalité n'est pas disponible en mode standard.
- Capture tous les artefacts : sur l'hôte et sur l'appareil, le mode personnalisé vous permet de capturer tous les artefacts de test. Cela peut ne pas être possible en mode standard.
- Environnement local plus cohérent et reproductible : en mode standard, les artefacts seront fournis séparément pour chaque test individuel, ce qui peut être bénéfique dans certaines circonstances. Cependant, votre environnement de test local peut s'écarter de la configuration d'origine car Device Farm gère différemment chaque test exécuté.

En revanche, le mode personnalisé vous permet de faire en sorte que votre environnement d'exécution des tests Device Farm soit toujours conforme à votre environnement de test local.

Les environnements personnalisés sont configurés à l'aide d'un fichier de spécification de test au format YAML (spécification de test). Device Farm fournit un fichier de spécifications de test par défaut pour chaque type de test pris en charge, qui peut être utilisé tel quel ou personnalisé ; des personnalisations telles que des filtres de test ou des fichiers de configuration peuvent être ajoutées aux spécifications de test. Les spécifications de test modifiées peuvent être enregistrées pour les futurs tests.

Pour plus d'informations, consultez la section [Téléchargement d'une spécification de test personnalisée à l'aide du AWS CLI](#) et [Création d'un test dans Device Farm](#)

Rubriques

- [Référence et syntaxe des spécifications de test](#)
- [Hôtes pour environnements de test personnalisés](#)
- [Accédez aux ressources AWS à l'aide d'un rôle d'exécution IAM](#)
- [Variables d'environnement pour les environnements de test personnalisés](#)
- [Bonnes pratiques pour l'exécution d'un environnement de test personnalisé](#)
- [Migration des tests d'un environnement de test standard vers un environnement de test personnalisé](#)
- [Extension des environnements de test personnalisés dans Device Farm](#)

Référence et syntaxe des spécifications de test

La spécification de test (spécification de test) est un fichier que vous utilisez pour définir des environnements de test personnalisés dans Device Farm.

Flux de travail des spécifications de test

La spécification de test Device Farm exécute les phases et leurs commandes dans un ordre prédéterminé, ce qui vous permet de personnaliser la façon dont votre environnement est préparé et exécuté. Lorsque chaque phase est exécutée, ses commandes sont exécutées dans l'ordre indiqué dans le fichier de spécifications de test. Les phases sont exécutées dans l'ordre suivant

1. `install`- C'est là que les actions telles que le téléchargement, l'installation et la configuration des outils doivent être définies.
2. `pre_test`- C'est là que les actions de pré-test, telles que le démarrage de processus en arrière-plan, doivent être définies.
3. `test`- C'est ici que la commande qui appelle votre test doit être définie.
4. `post_test`- C'est ici que doivent être définies toutes les tâches finales qui doivent être exécutées après la fin de votre test, telles que la génération de rapports de test et l'agrégation de fichiers d'artefacts.

Syntaxe des spécifications de test

Le schéma YAML d'un fichier de spécifications de test est le suivant :

```
version: 0.1

android_test_host: "string"
ios_test_host: "string"

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

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

version

(Obligatoire, numéro)

Reflète la version des spécifications de test prise en charge par Device Farm. Le numéro de version actuel est 0.1.

android_test_host

(Facultatif, chaîne)

L'hôte de test qui sera sélectionné pour les tests effectués sur les appareils Android. Ce champ est obligatoire pour les tests sur les appareils Android. Pour de plus amples informations, veuillez consulter [Hôtes de test disponibles pour les environnements de test personnalisés](#).

ios_test_host

(Facultatif, chaîne)

L'hôte de test qui sera sélectionné pour les tests effectués sur des appareils iOS. Ce champ est obligatoire pour les tests sur les appareils iOS dotés d'une version majeure supérieure à 26. Pour de plus amples informations, veuillez consulter [Hôtes de test disponibles pour les environnements de test personnalisés](#).

phases

Cette section contient des groupes de commandes exécutées lors d'un essai, chaque phase étant facultative. Les noms de phase de test autorisés sont : `installpre_test`, `test`, et `post_test`.

- `install`- Les dépendances par défaut pour les frameworks de test pris en charge par Device Farm sont déjà installées. Cette phase contient des commandes supplémentaires, le cas échéant, que Device Farm exécute pendant l'installation.
- `pre_test`- Les commandes, le cas échéant, exécutées avant votre test automatique.
- `test`- Les commandes exécutées lors de votre test automatique. Si une commande de la phase de test échoue (c'est-à-dire qu'elle renvoie un code de sortie différent de zéro), le test est marqué comme ayant échoué
- `post_test`- Les commandes, le cas échéant, exécutées après votre test automatique. Cela sera exécuté, que votre test réussisse ou non dans la `test` phase.

commands

(Facultatif, Liste [chaîne])

Une liste de chaînes à exécuter en tant que commande shell pendant la phase.

artifacts

(Facultatif, Liste [chaîne])

Device Farm collecte des artefacts tels que des rapports personnalisés, des fichiers journaux et des images à partir d'un emplacement spécifié ici. Les caractères génériques ne sont pas pris en charge dans le cadre d'un emplacement d'artefact. Par conséquent, vous devez spécifier un chemin d'accès valide pour chaque emplacement.

Ces artefacts de test sont disponibles pour chaque appareil de votre test. Pour plus d'informations sur la récupération de vos artefacts de test, consultez [Téléchargement d'artefacts dans un environnement de test personnalisé](#).

⚠ Important

Une spécification de test doit être mise en forme comme un fichier YAML valide. Si la mise en retrait ou l'espacement ne sont pas valides dans votre spécification de test, votre test peut échouer. Les tabulations ne sont pas autorisées dans les fichiers YAML. Vous pouvez utiliser un validateur YAML pour vérifier si votre spécification de test est un fichier YAML valide. Pour plus d'informations, consultez le [site web YAML](#).

Exemples de spécifications de test

Les exemples suivants présentent les spécifications de test qui peuvent être exécutées sur Device Farm.

Simple Demo

Voici un exemple de fichier de spécifications de test qui se connecte simplement en Hello world! tant qu'artefact d'exécution d'un test.

```
version: 0.1

android_test_host: amazon_linux_2
ios_test_host: macos_sequoia

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

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

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

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

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

Android Appium

Voici un exemple de fichier de spécifications de test qui configure un test Appium Java TestNG exécuté sur Android.

```
version: 0.1

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

phases:

  # The install phase contains commands for installing dependencies to run your
  # tests.
  # Certain frequently used dependencies are preinstalled on the test host to
  # accelerate and
  # simplify your test setup. To find these dependencies, versions supported and
  # additional
  # software installation please see:
  # https://docs.aws.amazon.com/devicefarm/latest/developerguide/custom-test-environments-hosts-software.html
  install:
    commands:
      # The Appium server is written using Node.js. In order to run your desired
      # version of Appium,
      # you first need to set up a Node.js environment that is compatible with your
      # version of Appium.
      - devicefarm-cli use node 20
      - node --version
```

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

# The Device Farm service periodically updates the preinstalled Appium
versions over time to
# incorporate the latest minor and patch versions for each major version. If
you wish to
# select a specific version of Appium, you can use NPM to install it.
# - npm install -g appium@2.19.0

# When running Android tests with Appium version 2, the uiautomator2 driver is
preinstalled using driver
# version 2.44.1 for Appium 2.5.1 If you want to install a different version
of the driver,
# you can use the Appium extension CLI to uninstall the existing uiautomator2
driver
# and install your desired version:
# - |-
#   if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "Android" ];
#   then
#       appium driver uninstall uiautomator2;
#       appium driver install uiautomator2@2.34.0;
#   fi;

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

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

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

```

# We recommend starting the Appium server process in the background using the
command below.
# The Appium server log will be written to the $DEVICEFARM_LOG_DIR directory.
# The environment variables passed as capabilities to the server will be
automatically assigned
# during your test run based on your test's specific device.
# For more information about which environment variables are set and how
they're set, please see
# https://docs.aws.amazon.com/devicefarm/latest/developerguide/custom-test-
environment-variables.html
- |-
  appium --base-path=$APPIUM_BASE_PATH --log-timestamp \
    --log-no-colors --relaxed-security --default-capabilities \
    '{"appium:deviceName": \"$DEVICEFARM_DEVICE_NAME\", \
    \"platformName\": \"$DEVICEFARM_DEVICE_PLATFORM_NAME\", \
    \"appium:udid\": \"$DEVICEFARM_DEVICE_UDID\", \
    \"appium:platformVersion\": \"$DEVICEFARM_DEVICE_OS_VERSION\", \
    \"appium:chromedriverExecutableDir\": \
    \"$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR\", \
    \"appium:automationName\": \"UiAutomator2\"}' \
    >> $DEVICEFARM_LOG_DIR/appium.log 2>&1 &

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

# The test phase contains commands for running your tests.
test:
  commands:
    # Your test package is downloaded and unpackaged into the
    $DEVICEFARM_TEST_PACKAGE_PATH directory.
    - echo "Navigate to test package directory"
    - cd $DEVICEFARM_TEST_PACKAGE_PATH
    - echo "Starting the Appium TestNG test"

```

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

To run your tests with a testng.xml file that is a part of your test package,

use the following commands instead:

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

The post-test phase contains commands that are run after your tests have completed.

If you need to run any commands to generating logs and reports on how your test performed,

we recommend adding them to this section.

post_test:

commands:

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

All files in these paths will be collected by Device Farm, with certain limits (see limit details

here: <https://docs.aws.amazon.com/devicefarm/latest/developerguide/limits.html#file-limits>).

These files will be available through the ListArtifacts API as your "Customer Artifacts".

artifacts:

By default, Device Farm will collect your artifacts from the \$DEVICEFARM_LOG_DIR directory.

- \$DEVICEFARM_LOG_DIR

Appium pour iOS

Voici un exemple de fichier de spécifications de test qui configure un test Appium Java TestNG exécuté sur iOS.

```
version: 0.1

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

phases:

  # The install phase contains commands for installing dependencies to run your
  # tests.
  # Certain frequently used dependencies are preinstalled on the test host to
  # accelerate and
  # simplify your test setup. To find these dependencies, versions supported and
  # additional
  # software installation please see:
  # https://docs.aws.amazon.com/devicefarm/latest/developerguide/custom-test-
  # environments-hosts-software.html
  install:
    commands:
      # The Appium server is written using Node.js. In order to run your desired
      # version of Appium,
      # you first need to set up a Node.js environment that is compatible with your
      # version of Appium.
      - devicefarm-cli use node 20
      - node --version

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

      # The Device Farm service periodically updates the preinstalled Appium
      # versions over time to
      # incorporate the latest minor and patch versions for each major version. If
      # you wish to
      # select a specific version of Appium, you can use NPM to install it.
      # - npm install -g appium@2.19.0

      # When running iOS tests with Appium version 2, the XCUITest driver is
      # preinstalled using driver
```

```

# version 9.10.5 for Appium 2.5.4. If you want to install a different version
of the driver,
# you can use the Appium extension CLI to uninstall the existing XCUITest
driver
# and install your desired version:
# - |-
#   if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "iOS" ];
#   then
#       appium driver uninstall xcuitest;
#       appium driver install xcuitest@10.0.0;
#   fi;

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

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

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

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

```

```

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

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

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

```



```

    \"appium:udid\": \"\$DEVICEFARM_DEVICE_UDID_FOR_APPIUM\", \
    \"appium:platformVersion\": \"\$DEVICEFARM_DEVICE_OS_VERSION\", \
    \"appium:derivedDataPath\": \"\$DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH\",
\
    \"appium:usePrebuiltWDA\": true, \
    \"appium:automationName\": \"XCUITest\"}" \
>> $DEVICEFARM_LOG_DIR/appium.log 2>&1 &

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

# The test phase contains commands for running your tests.
test:
    commands:
        # Your test package is downloaded and unpackaged into the
$DEVICEFARM_TEST_PACKAGE_PATH directory.
        - echo "Navigate to test package directory"
        - cd $DEVICEFARM_TEST_PACKAGE_PATH
        - echo "Starting the Appium TestNG test"

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

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

        # - echo "Unzipping the tests JAR file"

```

```

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

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

# Artifacts are a list of paths on the filesystem where you can store test output
and reports.
# All files in these paths will be collected by Device Farm, with certain limits
(see limit details
# here: https://docs.aws.amazon.com/devicefarm/latest/developerguide/limits.html#file-limits).
# These files will be available through the ListArtifacts API as your "Customer
Artifacts".
artifacts:
  # By default, Device Farm will collect your artifacts from the $DEVICEFARM_LOG_DIR
  directory.
  - $DEVICEFARM_LOG_DIR

```

Appium (Both Platforms)

Voici un exemple de fichier de spécifications de test qui configure un test Appium Java TestNG exécuté à la fois sur Android et iOS.

```

version: 0.1

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

phases:

  # The install phase contains commands for installing dependencies to run your
  tests.

```

```
# Certain frequently used dependencies are preinstalled on the test host to
accelerate and
# simplify your test setup. To find these dependencies, versions supported and
additional
# software installation please see:
# https://docs.aws.amazon.com/devicefarm/latest/developerguide/custom-test-
environments-hosts-software.html
install:
  commands:
    # The Appium server is written using Node.js. In order to run your desired
    version of Appium,
    # you first need to set up a Node.js environment that is compatible with your
    version of Appium.
    - devicefarm-cli use node 20
    - node --version

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

    # The Device Farm service periodically updates the preinstalled Appium
    versions over time to
    # incorporate the latest minor and patch versions for each major version. If
    you wish to
    # select a specific version of Appium, you can use NPM to install it.
    # - npm install -g appium@2.19.0

    # When running Android tests with Appium version 2, the uiautomator2 driver is
    preinstalled using driver
    # version 2.44.1 for Appium 2.5.1 If you want to install a different version
    of the driver,
    # you can use the Appium extension CLI to uninstall the existing uiautomator2
    driver
    # and install your desired version:
    # - |-
    #   if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "Android" ];
    #   then
    #     appium driver uninstall uiautomator2;
    #     appium driver install uiautomator2@2.34.0;
    #   fi;

    # When running iOS tests with Appium version 2, the XCUITest driver is
    preinstalled using driver
```

```

# version 9.10.5 for Appium 2.5.4. If you want to install a different version
of the driver,
# you can use the Appium extension CLI to uninstall the existing XCUITest
driver
# and install your desired version:
# - |-
#   if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "iOS" ];
#   then
#       appium driver uninstall xcuitest;
#       appium driver install xcuitest@10.0.0;
#   fi;

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

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

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

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

```

```

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

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

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

```

```

        \ "platformName\: \"\$DEVICEFARM_DEVICE_PLATFORM_NAME\", \
        \ "appium:udid\: \"\$DEVICEFARM_DEVICE_UDID\", \
        \ "appium:platformVersion\: \"\$DEVICEFARM_DEVICE_OS_VERSION\", \
        \ "appium:chromedriverExecutableDir\:
    \"\$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR\", \
        \ "appium:automationName\: \"UiAutomator2\"} \" \
    >> $DEVICEFARM_LOG_DIR/appium.log 2>&1 &
else
    appium --base-path=$APPIUM_BASE_PATH --log-timestamp \
        --log-no-colors --relaxed-security --default-capabilities \
        \"{ \"appium:deviceName\: \"\$DEVICEFARM_DEVICE_NAME\", \
        \ "platformName\: \"\$DEVICEFARM_DEVICE_PLATFORM_NAME\", \
        \ "appium:udid\: \"\$DEVICEFARM_DEVICE_UDID_FOR_APPIUM\", \
        \ "appium:platformVersion\: \"\$DEVICEFARM_DEVICE_OS_VERSION\", \
        \ "appium:derivedDataPath\: \"\$DEVICEFARM_WDA_DERIVED_DATA_PATH\", \
        \ "appium:usePrebuiltWDA\: true, \
        \ "appium:automationName\: \"XCUITest\"} \" \
    >> $DEVICEFARM_LOG_DIR/appium.log 2>&1 &
fi;

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

# The test phase contains commands for running your tests.
test:
    commands:
        # Your test package is downloaded and unpackaged into the
    $DEVICEFARM_TEST_PACKAGE_PATH directory.
        - echo "Navigate to test package directory"
        - cd $DEVICEFARM_TEST_PACKAGE_PATH
        - echo "Starting the Appium TestNG test"

    # The following command runs your Appium Java TestNG test.

```

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

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

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

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

# Artifacts are a list of paths on the filesystem where you can store test output
and reports.
# All files in these paths will be collected by Device Farm, with certain limits
(see limit details
# here: https://docs.aws.amazon.com/devicefarm/latest/developerguide/limits.html#file-limits).
# These files will be available through the ListArtifacts API as your "Customer
Artifacts".
artifacts:
  # By default, Device Farm will collect your artifacts from the $DEVICEFARM_LOG_DIR
  directory.
  - $DEVICEFARM_LOG_DIR
```

Hôtes pour environnements de test personnalisés

Device Farm prend en charge un ensemble de systèmes d'exploitation dotés de logiciels préconfigurés grâce à l'utilisation d'un environnement hôte de test. Pendant l'exécution du test, Device Farm utilise des instances gérées par Amazon (hôtes) qui se connectent dynamiquement à l'appareil sélectionné en cours de test. Cette instance est entièrement nettoyée et n'est pas réutilisée entre les exécutions. Elle est interrompue avec les artefacts générés une fois le test terminé.

Rubriques

- [Hôtes de test disponibles pour les environnements de test personnalisés](#)
- [Sélection d'un hôte de test pour les environnements de test personnalisés](#)
- [Logiciels pris en charge dans des environnements de test personnalisés](#)
- [Environnement de test pour appareils Android](#)
- [Environnement de test pour appareils iOS](#)

Hôtes de test disponibles pour les environnements de test personnalisés

Les hôtes de test sont entièrement gérés par Device Farm. Le tableau suivant répertorie les hôtes de test Device Farm actuellement disponibles et pris en charge pour les environnements de test personnalisés.

Plateforme d'appareils	Hôte du test	Système d'exploitation	Architecture (s)	Appareils pris en charge
Android	amazon_linux_2	Amazon Linux 2	x86_64	Android6 et plus
iOS	macos_sequoia	macOS Sequoia(version 15)	arm64	iOS15 à 26

Note

Device Farm ajoute régulièrement de nouveaux hôtes de test pour une plate-forme d'appareil afin de prendre en charge les nouvelles versions du système d'exploitation de l'appareil et

ses dépendances. Dans ce cas, les anciens hôtes de test pour la plate-forme d'appareil concernée sont sujets à la fin du support.

Version du système d'exploitation

Chaque hôte de test disponible utilise une version spécifique du système d'exploitation pris en charge par Device Farm à ce moment-là. Bien que nous essayions d'utiliser la dernière version du système d'exploitation, il se peut que ce ne soit pas la dernière version distribuée publiquement disponible. Device Farm mettra régulièrement à jour le système d'exploitation avec des mises à jour mineures et des correctifs de sécurité.

Pour connaître la version spécifique (y compris la version mineure) du système d'exploitation utilisée lors de votre exécution de test, vous pouvez ajouter l'extrait de code suivant à l'une des phases de votre fichier de spécifications de test.

Exemple

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

Sélection d'un hôte de test pour les environnements de test personnalisés

Vous pouvez spécifier l'hôte de test Android et iOS dans les `ios_test_host` variables appropriées `android_test_host` de votre [fichier de spécifications de test](#).

Si vous ne spécifiez pas de sélection d'hôte de test pour la plate-forme d'appareil donnée, les tests seront exécutés sur l'hôte de test que Device Farm a défini par défaut pour le périphérique et la configuration de test spécifiés.

⚠ Important

Lors des tests sur iOS 18 et versions antérieures, un hôte de test existant sera utilisé lorsqu'aucun hôte n'est sélectionné. Pour plus d'informations, consultez la rubrique sur le [Ancien hôte de test iOS](#).

À titre d'exemple, consultez l'extrait de code suivant :

Exemple

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

phases:
  # ...
```

Logiciels pris en charge dans des environnements de test personnalisés

Device Farm utilise des machines hôtes préinstallées avec de nombreuses bibliothèques logicielles nécessaires pour exécuter les frameworks de test pris en charge par notre service, fournissant ainsi un environnement de test prêt au lancement. Device Farm prend en charge plusieurs langues grâce à notre mécanisme de sélection de logiciels et mettra régulièrement à jour les versions des langues incluses dans l'environnement.

Pour tout autre logiciel requis, vous pouvez modifier le fichier de spécifications de test pour l'installer à partir de votre package de test, le télécharger depuis Internet ou accéder à des sources privées au sein de votre VPC (voir [VPC ENI](#) pour plus d'informations). Pour de plus amples informations, veuillez consulter [Exemples de spécifications de test](#).

Logiciel préconfiguré

Afin de faciliter le test des appareils sur chaque plate-forme, les outils suivants sont fournis sur l'hôte de test :

Outils	Plate-forme (s) d'appareils
Android SDK Build-Tools	Android

Outils	Plate-forme (s) d'appareils
Android SDK Platform-Tools(inclutadb)	Android
Xcode	iOS

Logiciel sélectionnable

Outre le logiciel préconfiguré sur l'hôte, Device Farm permet de sélectionner certaines versions des logiciels pris en charge via l'`devicefarm-cli` outillage.

Le tableau suivant répertorie les logiciels sélectionnables et les hôtes de test qui les contiennent.

Logiciel/Outil	Hôtes compatibles avec ce logiciel	Commande à utiliser dans votre spécification de test
Java 17	amazon_linux_2 macos_sequoia	<code>devicefarm-cli use java 17</code>
Java 11	amazon_linux_2 macos_sequoia	<code>devicefarm-cli use java 11</code>
Java 8	amazon_linux_2 macos_sequoia	<code>devicefarm-cli use java 8</code>
Node.js 20	amazon_linux_2 macos_sequoia	<code>devicefarm-cli use node 20</code>
Node.js 18	amazon_linux_2 macos_sequoia	<code>devicefarm-cli use node 18</code>
Node.js 16	amazon_linux_2	<code>devicefarm-cli use node 16</code>

Logiciel/Outil	Hôtes compatibles avec ce logiciel	Commande à utiliser dans votre spécification de test
Python 3.11	amazon_linux_2 macos_sequoia	<code>devicefarm-cli use python 3.11</code>
Python 3.10	amazon_linux_2 macos_sequoia	<code>devicefarm-cli use python 3.10</code>
Python 3.9	amazon_linux_2 macos_sequoia	<code>devicefarm-cli use python 3.9</code>
Python 3.8	amazon_linux_2	<code>devicefarm-cli use python 3.8</code>
Ruby 3.2	amazon_linux_2 macos_sequoia	<code>devicefarm-cli use ruby 3.2</code>
Ruby 2.7	amazon_linux_2	<code>devicefarm-cli use ruby 2.7</code>
Appium 3	amazon_linux_2	<code>devicefarm-cli use appium 3</code>
Appium 2	amazon_linux_2 macos_sequoia	<code>devicefarm-cli use appium 2</code>
Appium 1	amazon_linux_2	<code>devicefarm-cli use appium 1</code>
Xcode 26	macos_sequoia	<code>devicefarm-cli use xcode 26</code>
Xcode 16	macos_sequoia	<code>devicefarm-cli use xcode 16</code>

L'hôte de test inclut également des outils de support couramment utilisés pour chaque version du logiciel, tels que les gestionnaires de npm packages `pip` et (inclus respectivement avec Python et Node.js) et les dépendances (telles que le UIAutomator2 pilote Appium) pour des outils tels qu'Appium. Cela garantit que vous disposez des outils nécessaires pour travailler avec les frameworks de test pris en charge.

Utilisation de l'outil `devicefarm-cli` dans des environnements de test personnalisés

L'hôte de test utilise un outil de gestion des versions normalisé appelé `devicefarm-cli` à sélectionner les versions du logiciel. Cet outil est distinct de l'hôte de test Device Farm AWS CLI et n'est disponible que sur celui-ci. Avec `devicefarm-cli`, vous pouvez passer à n'importe quelle version logicielle préinstallée sur l'hôte de test. Cela fournit un moyen simple de gérer votre fichier de spécifications de test Device Farm au fil du temps et vous donne un mécanisme prévisible pour mettre à niveau les versions logicielles à l'avenir.

Important

Cet outil de ligne de commande n'est pas disponible sur les anciens hôtes iOS. Pour plus d'informations, consultez la rubrique sur le [Ancien hôte de test iOS](#).

L'extrait ci-dessous montre la help page de : `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.
```

Passons en revue quelques exemples d'utilisation de `devicefarm-cli`. Pour utiliser l'outil afin de modifier la version de Python de **3.10** à **3.9** dans votre fichier de spécifications de test, exécutez les commandes suivantes :

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

```
$ python --version
Python 3.9.17
```

Pour passer de la version d'Appium à **1 : 2**

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



Notez que lorsque vous sélectionnez une version logicielle, vous `devicefarm-cli` changez également les outils de support pour ces langages, tels que `pip` Python et `npm` NodeJS.

Pour plus d'informations sur le logiciel préinstallé sur l'hôte de test, consultez [Logiciels pris en charge dans des environnements de test personnalisés](#).

Environnement de test pour appareils Android

AWS Device Farm utilise les machines hôtes Amazon Elastic Compute Cloud (EC2) exécutant Amazon Linux 2 pour exécuter des tests Android. Lorsque vous planifiez un test, Device Farm alloue un hôte dédié à chaque appareil afin d'exécuter des tests de manière indépendante. Les machines hôtes s'arrêtent après le test, ainsi que tous les artefacts générés.

L'hôte Amazon Linux 2 offre plusieurs avantages :

- Tests plus rapides et plus fiables : par rapport à l'ancien hôte, le nouvel hôte de test améliore considérablement la vitesse des tests, notamment en réduisant les temps de démarrage des tests. L'hôte Amazon Linux 2 fait également preuve d'une stabilité et d'une fiabilité accrues lors des tests.
- Accès à distance amélioré pour les tests manuels : les mises à niveau vers le dernier hôte de test et les améliorations permettent de réduire la latence et d'améliorer les performances vidéo pour les tests manuels sur Android.
- Sélection de la version logicielle standard : Device Farm normalise désormais la prise en charge des principaux langages de programmation sur l'hôte de test ainsi que sur les versions du framework Appium. Pour les langages pris en charge (actuellement Java, Python, Node.js et Ruby)

et Appium, le nouvel hôte de test fournit des versions stables à long terme peu après le lancement. La gestion centralisée des versions via l'`devicefarm-cli` permet le développement de fichiers de spécifications de test avec une expérience cohérente dans tous les frameworks.

Rubriques

- [Plages d'adresses IP prises en charge pour l'environnement de test Amazon Linux 2 dans Device Farm](#)

Plages d'adresses IP prises en charge pour l'environnement de test Amazon Linux 2 dans Device Farm

Les clients ont souvent besoin de connaître la plage d'adresses IP d'où provient le trafic de Device Farm, notamment pour configurer leurs pare-feux et leurs paramètres de sécurité. Pour les hôtes de test Amazon EC2, la plage d'adresses IP couvre l'ensemble us-west-2 de la région. Pour les hôtes de test Amazon Linux 2, qui est l'option par défaut pour les nouvelles versions d'Android, les plages ont été limitées. Le trafic provient désormais d'un ensemble spécifique de passerelles NAT, limitant la plage d'adresses IP aux adresses suivantes :

Plages IP

44,236,137,143

52,13,151,244

52,35,189,191

54,201,250,26

Pour plus d'informations sur les environnements de test Android dans Device Farm, consultez [Environnement de test pour appareils Android](#).

Environnement de test pour appareils iOS

Device Farm utilise des instances de macOS (hôtes) gérées par Amazon qui se connectent dynamiquement à l'appareil iOS pendant le test. Chaque hôte est préconfiguré avec un logiciel qui permet de tester les appareils sur diverses plateformes de test populaires, telles que XCTest UI et Appium.

L'itération actuelle de l'hôte de test iOS a amélioré l'expérience de test par rapport aux versions précédentes, notamment :

- Expérience cohérente du système d'exploitation hôte et de l'outillage pour iOS 15 à iOS 26
Auparavant, l'hôte de test était déterminé par l'appareil utilisé, ce qui entraînait un environnement logiciel fragmenté lors de l'exécution sur plusieurs versions d'iOS. L'expérience actuelle permet une sélection simple des hôtes afin de créer un environnement cohérent sur tous les appareils. Cela permettra à la même version de macOS et aux mêmes outils (tels que Xcode) d'être disponibles sur chaque appareil iOS.
- Améliorations des performances pour les tests iOS 15 et 16 Grâce à une infrastructure mise à jour, le temps de configuration s'est considérablement amélioré pour les tests iOS 15 et 16.
- Versions logicielles standardisées sélectionnables pour les dépendances prises en charge Nous disposons désormais d'un système de sélection de `devicefarm-cli` logiciels sur les hôtes de test iOS et Android, qui vous permet de sélectionner votre version préférée de nos dépendances prises en charge. Pour les dépendances prises en charge (telles que Java, Python, Node.js, Ruby et Appium), les versions seront sélectionnables via la spécification de test. Pour avoir une idée du fonctionnement de cette fonctionnalité, consultez la rubrique sur [Logiciels pris en charge dans des environnements de test personnalisés](#).

Important

S'ils sont exécutés sur iOS 18 ou une version antérieure, vos tests s'exécuteront par défaut sur les anciens hôtes de test. Consultez la rubrique ci-dessous pour savoir comment migrer hors des hôtes existants.

Ancien hôte de test iOS

Pour les tests existants sur iOS 18 et versions antérieures, les anciens hôtes de test sont sélectionnés par défaut pour les environnements de test personnalisés. Le tableau suivant indique la version de l'hôte de test exécutée avec la version de l'appareil iOS.

Système d'exploitation	Architecture (s)	Par défaut pour les appareils
macOS Sonoma(version 14)	arm64	iOS 18
macOS Ventura(version 13)	arm64	iOS 17

Système d'exploitation	Architecture (s)	Par défaut pour les appareils
macOS Monterey(version 12)	x86_64	iOS 16et ci-dessous

Pour sélectionner les nouveaux hôtes de test, consultez la rubrique concernant [Migration de vos environnements de test personnalisés vers les nouveaux hôtes de test iOS](#).

Logiciels pris en charge pour les appareils iOS

Afin de prendre en charge les tests d'appareils iOS, les hôtes de test Device Farm pour appareils iOS sont préconfigurés avec Xcode et ses outils de ligne de commande associés. Pour les autres logiciels disponibles, veuillez consulter la rubrique concernant [Logiciels pris en charge dans des environnements de test personnalisés](#).

Migration de vos environnements de test personnalisés vers les nouveaux hôtes de test iOS

Pour migrer les tests existants de l'ancien hôte vers le nouvel hôte de test macOS, vous devez développer de nouveaux fichiers de spécifications de test basés sur vos fichiers de spécifications préexistants.

L'approche recommandée consiste à commencer par l'exemple de fichier de spécifications de test pour les types de test souhaités, puis à migrer les commandes pertinentes de votre ancien fichier de spécifications de test vers le nouveau. Cela vous permet de tirer parti des nouvelles fonctionnalités et des optimisations de l'exemple de spécification de test pour le nouvel hôte tout en réutilisant des extraits de code existant.

Rubriques

- [Tutoriel : Migration des fichiers de spécifications de test iOS avec la console](#)
- [Différences entre les nouveaux hôtes de test et les anciens](#)

Tutoriel : Migration des fichiers de spécifications de test iOS avec la console

Dans cet exemple, la console Device Farm sera utilisée pour intégrer une spécification de test d'appareil iOS existante afin d'utiliser le nouvel hôte de test.

Étape 1 : Création de nouveaux fichiers de spécifications de test avec la console

1. Connectez-vous à la [console AWS Device Farm](#).
2. Accédez au projet Device Farm contenant vos tests d'automatisation.
3. Téléchargez une copie des spécifications de test existantes que vous souhaitez utiliser.
 - a. Cliquez sur l'option « Paramètres du projet » et accédez à l'onglet Téléchargements.
 - b. Accédez au fichier de spécifications de test que vous souhaitez intégrer.
 - c. Cliquez sur le bouton Télécharger pour créer une copie locale de ce fichier.
4. Retournez à la page Projet et cliquez sur Créer une exécution.
5. Renseignez les options de l'assistant comme si vous deviez démarrer une nouvelle exécution, mais arrêtez-vous sur l'option Sélectionner les spécifications de test.
6. À l'aide de la spécification de test iOS sélectionnée par défaut, cliquez sur le bouton Créer une spécification de test.
7. Modifiez la spécification de test sélectionnée par défaut dans l'éditeur de texte.
 - a. S'il n'est pas déjà présent, modifiez le fichier de spécifications de test pour sélectionner le nouvel hôte en utilisant :

```
ios_test_host: macos_sequoia
```

- b. À partir de la copie de votre spécification de test téléchargée lors d'une étape précédente, passez en revue chacune phase d'elles.
 - c. Copiez les commandes des phases de l'ancienne spécification de test dans chaque phase respective de la nouvelle spécification de test, en ignorant les commandes liées à l'installation ou à la sélection de Java, Python, Node.js, Ruby, Appium ou Xcode.
8. Entrez un nouveau nom de fichier dans la zone de texte Enregistrer sous.
 9. Cliquez sur le bouton Enregistrer en tant que nouveau pour enregistrer vos modifications.

Pour un exemple de fichier de spécifications de test que vous pouvez utiliser comme référence, consultez l'exemple fourni dans [Exemples de spécifications de test](#).

Étape 2 : Sélection du logiciel préinstallé

Dans le nouvel hôte de test, les versions logicielles préinstallées sont sélectionnées à l'aide d'un nouvel outil de gestion des versions standardisé appelé `devicefarm-cli`. Cet outillage est

désormais l'approche recommandée pour utiliser les différents logiciels que nous fournissons sur les hôtes de test.

Par exemple, vous devez ajouter la ligne suivante pour utiliser un JDK 17 différent dans votre environnement de test :

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

Pour plus d'informations sur les logiciels pris en charge disponibles, veuillez consulter : [Logiciels pris en charge dans des environnements de test personnalisés](#).

Étape 3 : Utilisation d'Appium et de ses dépendances via l'outil de sélection de logiciels

Le nouvel hôte de test prend uniquement en charge Appium 2.x et versions ultérieures. Veuillez sélectionner explicitement la version d'Appium à l'aide de `devicefarm-cli`, tout en supprimant les outils existants tels que `avm`. Par exemple :

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

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

La version d'Appium sélectionnée par `devicefarm-cli` est préinstallée avec une version compatible du pilote XCUITest pour iOS.

De plus, vous devrez mettre à jour vos spécifications de test pour les utiliser au `DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V9` lieu de `DEVICEFARM_WDA_DERIVED_DATA_PATH`. La nouvelle variable d'environnement indique une version prédéfinie de WebDriverAgent 9.x, qui est la dernière version prise en charge pour les tests Appium 2.

Pour plus d'informations, consultez [Sélection d'une WebDriverAgent version pour les tests iOS](#) et [Variables d'environnement pour les tests Appium](#).

Différences entre les nouveaux hôtes de test et les anciens

Lorsque vous modifiez votre fichier de spécifications de test pour utiliser le nouvel hôte de test iOS et que vous transférez vos tests depuis l'ancien hôte de test, tenez compte de ces principales différences d'environnement :

- Versions Xcode : dans l'ancien environnement hôte de test, la version Xcode disponible était basée sur la version iOS de l'appareil utilisé pour les tests. Par exemple, les tests sur les appareils iOS 18 ont utilisé Xcode 16 dans l'ancien hôte, tandis que les tests sur iOS 17 ont utilisé Xcode 15. Dans le nouvel environnement hôte, tous les appareils peuvent accéder aux mêmes versions de Xcode, ce qui permet de disposer d'un environnement cohérent pour les tests sur des appareils dotés de versions différentes. Pour obtenir la liste des versions de Xcode actuellement disponibles, consultez [Logiciels pris en charge](#).
- Sélection des versions logicielles : Dans de nombreux cas, les versions logicielles par défaut ont changé. Par conséquent, si vous ne sélectionniez pas explicitement votre version logicielle sur l'ancien hôte de test, vous souhaitez peut-être la spécifier maintenant sur le nouvel hôte de test en utilisant [devicefarm-cli](#). Dans la grande majorité des cas d'utilisation, nous recommandons aux clients de sélectionner explicitement les versions des logiciels qu'ils utilisent. En sélectionnant une version logicielle avec `devicefarm-cli` vous bénéficierez d'une expérience prévisible et cohérente et recevrez de nombreux avertissements si Device Farm prévoit de supprimer cette version de l'hôte de test.

De plus, les outils de sélection de logiciels tels que `nvm`, `pyenv`, `avm`, et `rxm` ont été supprimés au profit du nouveau système de sélection de `devicefarm-cli` logiciels.

- Versions logicielles disponibles : de nombreuses versions de logiciels précédemment préinstallés ont été supprimées et de nombreuses nouvelles versions ont été ajoutées. Assurez-vous donc que lorsque vous utilisez le `devicefarm-cli` pour sélectionner les versions de vos logiciels, vous sélectionnez les versions figurant dans la [liste des versions prises en charge](#).
- La **libimobiledevice** suite d'outils a été supprimée au profit d'outils plus récents ou de première génération permettant de suivre les tests actuels des appareils iOS et les normes du secteur. Pour iOS 17 et versions ultérieures, vous pouvez migrer la plupart des commandes pour utiliser un outil Xcode similaire, appelé `devicectl`. Pour plus d'informations `devicectl`, vous pouvez exécuter `xcrun devicectl help` depuis une machine sur laquelle Xcode est installé.
- Les chemins de fichiers codés en dur dans votre ancien fichier de spécifications de test d'hôte en tant que chemins absolus ne fonctionneront probablement pas comme prévu sur le nouvel hôte de test, et ils ne sont généralement pas recommandés pour l'utilisation des fichiers de spécifications de test. Nous vous recommandons d'utiliser des chemins relatifs et des variables d'environnement pour tout le code des fichiers de spécifications de test. Pour plus d'informations, consultez la rubrique sur [Bonnes pratiques pour l'exécution d'un environnement de test personnalisé](#).
- Version et architecture du système d'exploitation : Les anciens hôtes de test utilisaient diverses versions de macOS et architectures de processeur en fonction du périphérique attribué. Par conséquent, les utilisateurs peuvent remarquer certaines différences dans les bibliothèques

système disponibles dans l'environnement. Pour plus d'informations sur la version précédente du système d'exploitation hôte, consultez [Ancien hôte de test iOS](#).

- Pour les utilisateurs d'Appium, la méthode de sélection WebDriverAgent a été remplacée par un préfixe de variable d'environnement use `DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V` au lieu de l'ancien préfixe. `DEVICEFARM_WDA_DERIVED_DATA_PATH_V` Pour plus d'informations sur la variable mise à jour, consultez [Variables d'environnement pour les tests Appium](#).
- Pour les utilisateurs d'Appium Java, le nouvel hôte de test ne contient aucun fichier JAR préinstallé dans son chemin de classe, alors que l'hôte précédent en contenait un pour le framework TestNG (via une variable d'environnement). `$DEVICEFARM_TESTNG_JAR` Nous recommandons aux clients d'intégrer les fichiers JAR nécessaires à leurs frameworks de test dans leur package de test et de supprimer les instances de la `$DEVICEFARM_TESTNG_JAR` variable de leurs fichiers de spécifications de test.

Nous vous recommandons de contacter l'équipe de service par le biais d'un dossier d'assistance si vous avez des commentaires ou des questions sur les différences entre les hôtes de test du point de vue logiciel.

Accédez aux ressources AWS à l'aide d'un rôle d'exécution IAM

Device Farm permet de spécifier un rôle IAM qui sera assumé par l'environnement d'exécution de test personnalisé lors de l'exécution du test. Cette fonctionnalité permet à vos tests d'accéder en toute sécurité aux ressources AWS de votre compte, telles que les compartiments Amazon S3, les tables DynamoDB ou d'autres services AWS dont dépend votre application.

Rubriques

- [Aperçu](#)
- [Exigences relatives au rôle IAM](#)
- [Configuration d'un rôle d'exécution IAM](#)
- [Bonnes pratiques](#)
- [Résolution des problèmes](#)

Aperçu

Lorsque vous spécifiez un rôle d'exécution IAM, Device Farm assume ce rôle lors de l'exécution des tests, ce qui permet à vos tests d'interagir avec les services AWS en utilisant les autorisations définies dans le rôle.

Les cas d'utilisation courants des rôles d'exécution IAM incluent :

- Accès aux données de test stockées dans des compartiments Amazon S3
- Transférer des artefacts de test vers des compartiments Amazon S3
- Récupération de la configuration d'une application depuis AWS AppConfig
- Rédaction de journaux de test et de statistiques pour Amazon CloudWatch
- Envoi de résultats de test ou de messages d'état aux files d'attente Amazon SQS
- Appel de fonctions AWS Lambda dans le cadre de flux de travail de test

Exigences relatives au rôle IAM

Pour utiliser un rôle d'exécution IAM avec Device Farm, votre rôle doit répondre aux exigences suivantes :

- Relation de confiance : le responsable du service Device Farm doit être digne de confiance pour assumer ce rôle. La politique de confiance doit inclure `devicefarm.amazonaws.com` le fait d'être une entité de confiance.
- Autorisations : le rôle doit disposer des autorisations nécessaires pour accéder aux ressources AWS dont vos tests ont besoin.
- Durée de session : la durée maximale de session du rôle doit être au moins aussi longue que le délai d'expiration des tâches défini dans votre projet Device Farm. Par défaut, les projets Device Farm ont un délai d'expiration de 150 minutes. Votre rôle doit donc prendre en charge une durée de session d'au moins 150 minutes.
- Même exigence de compte : le rôle IAM doit figurer sur le même compte AWS que celui utilisé pour appeler Device Farm. L'hypothèse de rôles entre comptes n'est pas prise en charge.
- PassRole autorisation : l'appelant doit être autorisé à transmettre le rôle IAM par une politique autorisant `iam:PassRole` sur le rôle d'exécution spécifié.

Exemple de politique d'approbation

L'exemple suivant montre une politique de confiance qui permet à Device Farm d'assumer votre rôle d'exécution. Cette politique de confiance ne doit être attachée qu'au rôle IAM spécifique que vous avez l'intention d'utiliser avec Device Farm, et non aux autres rôles de votre compte :

Exemple

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

Exemple de politique d'autorisations

L'exemple suivant montre une politique d'autorisation qui accorde l'accès aux services AWS courants utilisés lors des tests :

Example

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


Configuration d'un rôle d'exécution IAM

Vous pouvez spécifier un rôle d'exécution IAM au niveau du projet ou pour des tests individuels. Lorsqu'elles sont configurées au niveau du projet, toutes les exécutions au sein de ce projet hériteront du rôle d'exécution. Un rôle d'exécution configuré lors d'une exécution remplacera tout rôle configuré sur son projet parent.

Pour obtenir des instructions détaillées sur la configuration des rôles d'exécution, voir :

- [Création d'un projet dans AWS Device Farm](#)- pour configurer les rôles d'exécution au niveau du projet
- [Création d'un test dans Device Farm](#)- pour configurer les rôles d'exécution pour des exécutions individuelles

Vous pouvez également configurer les rôles d'exécution à l'aide de l'API Device Farm. Pour plus d'informations, consultez le [Device Farm API Reference](#).

Bonnes pratiques

Suivez ces bonnes pratiques lors de la configuration des rôles d'exécution IAM pour vos tests Device Farm :

- Principe du moindre privilège : accordez uniquement les autorisations minimales nécessaires au bon fonctionnement de vos tests. Évitez d'utiliser des autorisations trop larges, telles que * des actions ou des ressources.
- Utiliser des autorisations spécifiques aux ressources : dans la mesure du possible, limitez les autorisations à des ressources spécifiques (par exemple, des compartiments S3 ou des tables DynamoDB spécifiques) plutôt qu'à toutes les ressources d'un type.
- Ressources de test et de production distinctes : utilisez des ressources et des rôles de test dédiés pour éviter d'affecter accidentellement les systèmes de production pendant les tests.
- Révision régulière des rôles : passez régulièrement en revue et mettez à jour vos rôles d'exécution pour vous assurer qu'ils répondent toujours à vos besoins de test et respectent les meilleures pratiques en matière de sécurité.
- Utiliser des clés de condition : pensez à utiliser des clés de condition IAM pour limiter davantage le moment et la manière dont le rôle peut être utilisé.

Résolution des problèmes

Si vous rencontrez des problèmes avec les rôles d'exécution IAM, vérifiez les points suivants :

- Relation de confiance : vérifiez que la politique de confiance du `devicefarm.amazonaws.com` rôle inclut un service de confiance.
- Autorisations : vérifiez que le rôle dispose des autorisations nécessaires pour les services AWS auxquels vos tests tentent d'accéder.
- Journaux de test : consultez les journaux d'exécution des tests pour détecter les messages d'erreur spécifiques liés aux appels d'API AWS ou aux refus d'autorisation.

Variables d'environnement pour les environnements de test personnalisés

Device Farm configure dynamiquement plusieurs variables d'environnement à utiliser dans le cadre de l'exécution de votre environnement de test personnalisé.

Rubriques

- [Variables d'environnement personnalisées](#)
- [Variables d'environnement communes](#)
- [Variables d'environnement pour les tests Appium](#)
- [Variables d'environnement pour les XCUITest tests](#)

Variables d'environnement personnalisées

Device Farm prend en charge la configuration de paires clé-valeur appliquées en tant que variables d'environnement sur l'hôte de test. Elles peuvent être configurées sur un projet Device Farm ou lors de la création d'une exécution ; toutes les variables configurées lors d'une exécution remplaceront celles qui peuvent être configurées sur son projet parent. Les restrictions suivantes s'appliquent :

- Les variables d'environnement personnalisées ne sont pas prises en charge sur les anciens hôtes de test iOS. Pour de plus amples informations, veuillez consulter [Ancien hôte de test iOS](#).
- Les noms de variables commençant par `$DEVICEFARM_` sont réservés à un usage interne du service.

- Les variables d'environnement personnalisées ne peuvent pas être utilisées pour configurer la sélection de calcul de l'hôte de test dans votre spécification de test.

Variables d'environnement communes

Cette section décrit les variables d'environnement communes à tous les tests de Device Farm.

\$DEVICEFARM_DEVICE_NAME

L'appareil sur lequel vos tests sont exécutés. Il représente l'identifiant unique de l'appareil (UDID).

\$DEVICEFARM_DEVICE_UDID

Identifiant unique de l'appareil.

\$DEVICEFARM_DEVICE_PLATFORM_NAME

Le nom de la plateforme de l'appareil. C'est l'un Android ou l'autre iOS.

\$DEVICEFARM_DEVICE_OS_VERSION

Version du système d'exploitation de l'appareil.

\$DEVICEFARM_APP_PATH

(tests d'applications mobiles)

Chemin de l'application mobile sur la machine hôte où les tests sont en cours d'exécution. Cette variable n'est pas disponible lors des tests Web.

\$DEVICEFARM_LOG_DIR

Le chemin d'accès au répertoire par défaut dans lequel les journaux des clients, les artefacts et les autres fichiers souhaités seront stockés pour être récupérés ultérieurement. À l'aide d'un [exemple de spécification de test](#), les fichiers de ce répertoire sont archivés dans un fichier ZIP et mis à disposition sous forme d'artefact après votre test.

\$DEVICEFARM_SCREENSHOT_PATH

Chemin des captures d'écran, le cas échéant, capturées au cours du test.

\$DEVICEFARM_PROJECT_ARN

L'ARN du projet parent de la tâche.

\$DEVICEFARM_RUN_ARN

L'ARN de l'exécution parent de la tâche.

\$DEVICEFARM_DEVICE_ARN

L'ARN de l'appareil testé.

\$DEVICEFARM_TOTAL_JOBS

Le nombre total de tâches associées à l'exécution de son Device Farm parent.

\$DEVICEFARM_JOB_NUMBER

Le numéro de ce job se trouve à l'intérieur \$DEVICEFARM_TOTAL_JOBS. Par exemple, une exécution peut contenir 5 tâches, chacune ayant un caractère unique \$DEVICEFARM_JOB_NUMBER compris entre 0 et 4.

\$AWS_REGION

Région AWS Le service le configurera pour qu'il corresponde à la région dans laquelle se trouve l'appareil testé. Elle peut être remplacée par une variable d'environnement personnalisée si nécessaire.

\$ANDROID_HOME

(Android uniquement)

Le chemin d'accès au répertoire d'installation du SDK Android.

Variables d'environnement pour les tests Appium

Cette section décrit les variables d'environnement utilisées par n'importe quel test Appium dans un environnement de test personnalisé dans Device Farm.

\$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR

(Android uniquement)

L'emplacement d'un répertoire contenant les ChromeDriver exécutables nécessaires à une utilisation dans les tests Web et hybrides d'Appium.

\$DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V<N>

(iOS uniquement)

Le chemin de données dérivé d'une version de Device Farm WebDriverAgent conçue pour fonctionner sur Device Farm. La numérotation de la variable correspondra à la version principale du WebDriverAgent. À titre d'exemple, `DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V9` indiquera la WebDriverAgent version a de 9.x. Pour de plus amples informations, veuillez consulter [Sélection d'une WebDriverAgent version pour les tests iOS](#).

 Note

Les variables

d'`$DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V<N>`environnement ne sont présentes que sur les hôtes iOS non existants. Pour de plus amples informations, veuillez consulter [Ancien hôte de test iOS](#).

`$DEVICEFARM_WDA_DERIVED_DATA_PATH_V9`

(iOS uniquement, obsolète)

Le chemin de données dérivé d'une version de Device Farm WebDriverAgent conçue pour fonctionner sur Device Farm. Reportez-vous

`$DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V<N>` au schéma de dénomination de remplacement.

Variables d'environnement pour les XCUITest tests

Cette section décrit les variables d'environnement utilisées par le XCUITest test dans un environnement de test personnalisé dans Device Farm.

`$DEVICEFARM_XCUITESTRUN_FILE`

Le chemin d'accès au `.xctest`un fichier Device Farm. Il est généré à partir de vos packages d'application et de test.

`$DEVICEFARM_DERIVED_DATA_PATH`

Chemin attendu de la sortie `xcodebuild` de Device Farm.

`$DEVICEFARM_XCTEST_BUILD_DIRECTORY`

Chemin du contenu décompressé du fichier du package de test.

Bonnes pratiques pour l'exécution d'un environnement de test personnalisé

Les rubriques suivantes présentent les meilleures pratiques recommandées pour l'utilisation de l'exécution de tests personnalisés avec Device Farm.

Configuration d'exécution

- Utilisez le logiciel géré par Device Farm et les fonctionnalités de l'API pour exécuter la configuration dans la mesure du possible, au lieu d'appliquer des configurations similaires via des commandes shell dans le fichier de spécifications de test. Cela inclut la configuration de l'hôte de test et du périphérique, car elle sera plus durable et cohérente entre les hôtes de test et les appareils.

Device Farm vous encourage à personnaliser votre fichier de spécifications de test autant que nécessaire pour exécuter vos tests, mais le fichier de spécifications de test peut devenir difficile à gérer au fil du temps à mesure que des commandes plus personnalisées y sont ajoutées. Utilisation du logiciel géré par Device Farm (via des outils tels que `devicefarm-cli` les outils disponibles par défaut dans le \$PATH) et utilisation de fonctionnalités gérées (telles que le paramètre de [deviceProxy](#) demande) pour simplifier le fichier de spécifications de test en transférant la responsabilité de la maintenance à Device Farm elle-même.

Spécifications de test et code du package de test

- N'utilisez pas de chemins absolus et ne vous fiez pas à des versions mineures spécifiques dans votre fichier de spécifications de test ou le code de votre package de test. Device Farm applique des mises à jour de routine à l'hôte de test sélectionné et aux versions logicielles incluses. L'utilisation de chemins spécifiques ou absolus (par exemple `/usr/local/bin/python` au lieu de `python`) ou l'exigence de versions mineures spécifiques (par exemple `Node.js 20.3.1` au lieu de simplement `20`) peuvent empêcher vos tests de localiser le fichier exécutable/requis.

Dans le cadre de l'exécution des tests personnalisés, Device Farm définit diverses variables d'environnement ainsi que la \$PATH variable afin de garantir une expérience cohérente dans nos environnements dynamiques. Pour plus d'informations, consultez [Variables d'environnement pour les environnements de test personnalisés](#) et [Logiciels pris en charge dans des environnements de test personnalisés](#).

- Enregistrez les fichiers générés ou copiés dans le répertoire temporaire pendant le test. Aujourd'hui, nous nous assurons que le répertoire temporaire (/tmp) sera accessible à l'utilisateur pendant l'exécution du test (en plus des répertoires gérés, tels que le `$DEVICEFARM_LOG_DIR`). Les autres répertoires auxquels l'utilisateur a accès peuvent changer au fil du temps en fonction des besoins du service ou du système d'exploitation utilisé.
- Enregistrez vos journaux d'exécution des tests dans `$DEVICEFARM_LOG_DIR`. Il s'agit du répertoire d'artefacts par défaut fourni à votre exécution pour y ajouter des journaux d'exécution ou des artefacts. Les [exemples de spécifications de test](#) que nous fournissons utilisent chacun ce répertoire pour les artefacts par défaut.
- Assurez-vous que vos commandes renvoient un code différent de zéro en cas d'échec pendant la test phase de votre spécification de test. Nous déterminons si votre exécution a échoué en vérifiant la présence d'un code de sortie différent de zéro pour chaque commande shell invoquée pendant la test phase. Vous devez vous assurer que votre logique ou votre infrastructure de test renverra un code de sortie différent de zéro pour tous les scénarios souhaités, qui peuvent nécessiter une configuration supplémentaire.

Par exemple, certains frameworks de test (tels que JUnit5) ne considèrent pas l'exécution d'un test comme un échec, ce qui entraînera la détection d'une exécution réussie de vos tests même si rien n'a été exécuté. À JUnit5 titre d'exemple, vous devez spécifier l'option de ligne de commande pour vous `--fail-if-no-tests` assurer que ce scénario se termine avec un code de sortie différent de zéro.

- Vérifiez la compatibilité du logiciel avec la version du système d'exploitation de l'appareil et la version de l'hôte de test que vous utiliserez pour le test. Par exemple, certaines fonctionnalités des frameworks logiciels de test (par exemple : Appium) peuvent ne pas fonctionner comme prévu sur toutes les versions du système d'exploitation de l'appareil testé.

Sécurité

- Évitez de stocker ou de consigner des variables sensibles (comme les clés AWS) dans votre fichier de spécifications de test. Les fichiers de spécifications de test, les scripts générés par les spécifications de test et les journaux des scripts de spécifications de test sont tous fournis sous forme d'artefacts téléchargeables à la fin de l'exécution du test. Cela peut entraîner la divulgation involontaire de secrets à d'autres utilisateurs de votre compte ayant un accès en lecture à votre test.

Migration des tests d'un environnement de test standard vers un environnement de test personnalisé

Vous pouvez passer d'un mode d'exécution de test standard à un mode d'exécution personnalisé dans AWS Device Farm. La migration implique principalement deux formes d'exécution différentes :

1. Mode standard : ce mode d'exécution des tests est principalement conçu pour fournir aux clients des rapports granulaires et un environnement entièrement géré.
2. Mode personnalisé : ce mode d'exécution des tests est conçu pour différents cas d'utilisation nécessitant des tests plus rapides, la capacité de levage et de décalage pour atteindre la parité avec leur environnement local, ainsi que la diffusion vidéo en direct.

Pour plus d'informations sur les modes standard et personnalisé de Device Farm, reportez-vous [Environnements de test dans AWS Device Farm](#) aux sections et [Environnements de test personnalisés dans AWS Device Farm](#).

Considérations relatives à la migration

Cette section répertorie certains des principaux cas d'utilisation à prendre en compte lors de la migration vers le mode personnalisé :

1. Rapidité : dans le mode d'exécution standard, Device Farm analyse les métadonnées des tests que vous avez empaquetés et téléchargés à l'aide des instructions d'empaquetage correspondant à votre framework spécifique. L'analyse détecte le nombre de tests dans votre package. Device Farm exécute ensuite chaque test séparément et présente les journaux, les vidéos et les autres artefacts des résultats individuellement pour chaque test. Cependant, cela augmente régulièrement le temps total d'exécution des end-to-end tests, car il y a le pré-traitement et le post-traitement des tests et des artefacts de résultats du côté du service.

En revanche, le mode d'exécution personnalisé n'analyse pas votre package de test ; cela signifie qu'il n'y a aucun prétraitement et un post-traitement minimal pour les tests ou les artefacts de résultats. Cela se traduit par des temps end-to-end d'exécution totaux proches de ceux de votre configuration locale. Les tests sont exécutés dans le même format que s'ils étaient exécutés sur votre ou vos machines locales. Les résultats des tests sont identiques à ceux que vous obtenez localement et peuvent être téléchargés à la fin de l'exécution de la tâche.

2. Personnalisation ou flexibilité : le mode d'exécution standard analyse votre package de test pour détecter le nombre de tests, puis exécute chaque test séparément. Notez qu'il n'y a aucune

garantie que les tests s'exécuteront dans l'ordre que vous avez spécifié. Par conséquent, les tests nécessitant une séquence d'exécution particulière peuvent ne pas fonctionner comme prévu. En outre, il n'existe aucun moyen de personnaliser l'environnement de la machine hôte ou de transmettre les fichiers de configuration qui peuvent être nécessaires pour exécuter vos tests d'une certaine manière.

En revanche, le mode personnalisé vous permet de configurer l'environnement de la machine hôte, notamment d'installer des logiciels supplémentaires, de transmettre des filtres à vos tests, de transmettre des fichiers de configuration et de contrôler la configuration d'exécution des tests. Pour ce faire, il utilise un fichier yaml (également appelé fichier testspec) que vous pouvez modifier en y ajoutant des commandes shell. Ce fichier yaml est converti en un script shell qui est exécuté sur la machine hôte de test. Vous pouvez enregistrer plusieurs fichiers yaml et en choisir un dynamiquement selon vos besoins lorsque vous planifiez une exécution.

3. Vidéo en direct et journalisation : les modes d'exécution standard et personnalisé vous fournissent des vidéos et des journaux pour vos tests. Cependant, en mode standard, vous n'obtenez la vidéo et les journaux prédéfinis de vos tests qu'une fois ceux-ci terminés.

En revanche, le mode personnalisé vous permet de diffuser en direct la vidéo et les journaux de vos tests côté client. De plus, vous pouvez télécharger la vidéo et d'autres artefacts à la fin du ou des tests.

Tip

Si votre cas d'utilisation implique au moins l'un des facteurs ci-dessus, nous vous recommandons vivement de passer au mode d'exécution personnalisé.

Étapes de la migration

Pour passer du mode standard au mode personnalisé, procédez comme suit :

1. Connectez-vous à la console Device Farm AWS Management Console et ouvrez-la à l'adresse <https://console.aws.amazon.com/devicefarm/>.
2. Choisissez votre projet, puis lancez une nouvelle opération d'automatisation.
3. Téléchargez votre application (ou sélectionnez web app), choisissez votre type de framework de test, téléchargez votre package de test, puis sous le Choose your execution environment paramètre, choisissez l'option pour Run your test in a custom environment.

4. Par défaut, l'exemple de fichier de spécifications de test de Device Farm s'affiche pour que vous puissiez le consulter et le modifier. Ce fichier d'exemple peut être utilisé comme point de départ pour essayer vos tests en [mode environnement personnalisé](#). Ensuite, une fois que vous avez vérifié que vos tests fonctionnent correctement depuis la console, vous pouvez modifier n'importe laquelle de vos intégrations d'API, de CLI ou de pipeline avec Device Farm pour utiliser ce fichier de spécifications de test en tant que paramètre lors de la planification des tests. Pour plus d'informations sur la façon d'ajouter un fichier de spécifications de test en tant que paramètre pour vos exécutions, consultez la section sur les `testSpecArn` paramètres de l'`ScheduleRunAPI` dans notre [guide des API](#).

Cadre Appium

Dans un environnement de test personnalisé, Device Farm n'insère ni ne remplace aucune fonctionnalité d'Appium dans vos tests de framework Appium. Vous devez spécifier les fonctionnalités Appium de votre test dans le fichier YAML de spécification de test ou dans votre code de test.

Instrumentation Android

Vous n'avez pas besoin d'effectuer de modifications pour déplacer vos tests d'instrumentation Android vers un environnement de test personnalisé.

iOS XCUITest

Il n'est pas nécessaire d'apporter des modifications pour déplacer vos XCUITest tests iOS vers un environnement de test personnalisé.

Extension des environnements de test personnalisés dans Device Farm

AWS Device Farm permet de configurer un environnement personnalisé pour les tests automatisés (mode personnalisé), ce qui est l'approche recommandée pour tous les utilisateurs de Device Farm. Le mode personnalisé Device Farm vous permet d'exécuter bien plus que votre suite de tests. Dans cette section, vous découvrirez comment étendre votre suite de tests et optimiser vos tests.

Pour plus d'informations sur les environnements de test personnalisés dans Device Farm, consultez [Environnements de test personnalisés dans AWS Device Farm](#).

Rubriques

- [Définition du code PIN d'un appareil lors de l'exécution de tests dans Device Farm](#)
- [Accélérer les tests basés sur Appium dans Device Farm grâce aux fonctionnalités souhaitées](#)
- [Utilisation de webhooks et autres APIs après l'exécution de vos tests dans Device Farm](#)
- [Ajouter des fichiers supplémentaires à votre package de test dans Device Farm](#)

Définition du code PIN d'un appareil lors de l'exécution de tests dans Device Farm

Certaines applications nécessitent que vous définissiez un code PIN sur l'appareil. Device Farm ne prend pas en charge la définition native d'un code PIN sur les appareils. Cela est toutefois possible avec les mises en garde suivantes :

- L'appareil doit fonctionner sous Android 8 ou une version ultérieure.
- Le code PIN doit être retiré une fois le test terminé.

Pour définir le code PIN lors de vos tests, utilisez les `post_test` phases `pre_test` et pour définir et supprimer le code PIN, comme indiqué ci-dessous :

```
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"
```

Lorsque votre suite de tests démarre, le code PIN 1234 est défini. Une fois votre suite de tests terminée, le code PIN est supprimé.

Warning

Si vous ne supprimez pas le code PIN de l'appareil une fois le test terminé, l'appareil et votre compte seront mis en quarantaine.

Pour découvrir d'autres moyens d'étendre votre suite de tests et d'optimiser vos tests, consultez [Extension des environnements de test personnalisés dans Device Farm](#).

Accélérer les tests basés sur Appium dans Device Farm grâce aux fonctionnalités souhaitées

Lorsque vous utilisez Appium, vous constaterez peut-être que la suite de tests en mode standard est très lente. Cela est dû au fait que Device Farm applique les paramètres par défaut et ne fait aucune hypothèse quant à la manière dont vous souhaitez utiliser l'environnement Appium. Bien que ces valeurs par défaut soient basées sur les meilleures pratiques du secteur, elles peuvent ne pas s'appliquer à votre situation. Pour affiner les paramètres du serveur Appium, vous pouvez ajuster les fonctionnalités Appium par défaut dans vos spécifications de test. Par exemple, ce qui suit définit la `usePrebuildWDA` fonctionnalité d'`true` une suite de tests iOS pour accélérer l'heure de démarrage initiale :

```
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 &
```

Les fonctionnalités d'Appium doivent être une structure JSON échappée au shell et entre guillemets.

Les fonctionnalités Appium suivantes sont des sources courantes d'amélioration des performances :

`noReset` et `fullReset`

Ces deux fonctionnalités, qui s'excluent mutuellement, décrivent le comportement d'Appium une fois chaque session terminée. Lorsqu'il `noReset` est défini sur `true`, le serveur Appium ne supprime pas les données de votre application lorsqu'une session Appium se termine, n'effectuant en fait aucun nettoyage. `fullReset` désinstalle et efface toutes les données d'application de

l'appareil après la fermeture de la session. Pour plus d'informations, consultez [Reset Strategies](#) dans la documentation d'Appium.

`ignoreUnimportantViews(Android uniquement)`

Demande à Appium de compresser la hiérarchie de l'interface utilisateur Android uniquement pour les vues pertinentes pour le test, ce qui accélère la recherche de certains éléments. Cependant, cela peut perturber certaines suites de tests XPath basées sur des données car la hiérarchie de la disposition de l'interface utilisateur a été modifiée.

`skipUnlock(Android uniquement)`

Informe Appium qu'aucun code PIN n'est actuellement défini, ce qui accélère les tests après un événement de fermeture d'écran ou un autre événement de verrouillage.

`webDriverAgentUrl(iOS uniquement)`

Demande à Appium de supposer qu'une dépendance iOS essentielle est déjà en cours d'exécution et disponible pour accepter les requêtes HTTP à l'URL spécifiée. `webdriverAgent` S'il `webdriverAgent` n'est pas déjà opérationnel, Appium peut mettre un certain temps au début d'une suite de tests pour démarrer le `webdriverAgent` Si vous démarrez `webdriverAgent` vous-même et que vous le configurez `webdriverAgentUrl` au `http://localhost:8100` démarrage d'Appium, vous pouvez démarrer votre suite de tests plus rapidement. Notez que cette fonctionnalité ne doit jamais être utilisée en même temps que la `useNewWDA` fonctionnalité.

Vous pouvez utiliser le code suivant pour commencer à `webdriverAgent` partir de votre fichier de spécifications de test sur le port local de l'appareil `8100`, puis le transférer vers le port local de l'hôte de test `8100` (cela vous permet `webdriverAgentUrl` de définir la valeur sur `http://localhost:8100`). Ce code doit être exécuté pendant la phase d'installation une fois que tout code de configuration d'Appium et des variables d'`webdriverAgent` environnement a été défini :

```
# 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 &
```

Vous pouvez ensuite ajouter le code suivant à votre fichier de spécifications de test pour vous assurer que le `webDriverAgent` démarrage est réussi. Ce code doit être exécuté à la fin de la phase de pré-test après avoir vérifié le bon démarrage d'Appium :

```
# 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;
```

Pour plus d'informations sur les fonctionnalités prises en charge par Appium, consultez la section [Capacités souhaitées d'Appium](#) dans la documentation d'Appium.

Pour découvrir d'autres moyens d'étendre votre suite de tests et d'optimiser vos tests, consultez [Extension des environnements de test personnalisés dans Device Farm](#).

Utilisation de webhooks et autres APIs après l'exécution de vos tests dans Device Farm

Device Farm peut appeler un webhook une fois que chaque suite de tests a fini d'être utilisée. curl La procédure à suivre varie en fonction de la destination et du formatage. Pour votre webhook

spécifique, consultez la documentation de ce webhook. L'exemple suivant publie un message chaque fois qu'une suite de tests est terminée sur un webhook Slack :

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

Pour plus d'informations sur l'utilisation des webhooks avec Slack, consultez la section [Envoyer votre premier message Slack à l'aide de Webhook dans le guide de référence de l'API Slack](#).

Pour découvrir d'autres moyens d'étendre votre suite de tests et d'optimiser vos tests, consultez [Extension des environnements de test personnalisés dans Device Farm](#).

Vous n'êtes pas limité à l'utilisation curl pour appeler des webhooks. Les packages de test peuvent inclure des scripts et des outils supplémentaires, à condition qu'ils soient compatibles avec l'environnement d'exécution de Device Farm. Par exemple, votre package de test peut inclure des scripts auxiliaires qui envoient des requêtes à d'autres APIs. Assurez-vous que tous les packages requis sont installés conformément aux exigences de votre suite de tests. Pour ajouter un script qui s'exécute une fois votre suite de tests terminée, incluez-le dans votre package de test et ajoutez ce qui suit à votre spécification de test :

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

Note

La maintenance des clés d'API ou des autres jetons d'authentification utilisés dans votre package de test est de votre responsabilité. Nous vous recommandons de garder toute forme d'identification de sécurité hors du contrôle de la source, d'utiliser des informations d'identification avec le moins de privilèges possibles et d'utiliser des jetons révocables de courte durée dans la mesure du possible. Pour vérifier les exigences de sécurité, consultez la documentation du tiers APIs que vous utilisez.

Si vous prévoyez d'utiliser AWS des services dans le cadre de votre suite d'exécution de tests, vous devez utiliser des informations d'identification temporaires IAM, générées en dehors de votre

suite de tests et incluses dans votre package de test. Ces informations d'identification doivent avoir le moins d'autorisations accordées et avoir une durée de vie la plus courte possible. Pour plus d'informations sur la création d'informations d'identification temporaires, consultez la section [Demande d'informations d'identification de sécurité temporaires](#) dans le guide de l'utilisateur IAM.

Pour découvrir d'autres moyens d'étendre votre suite de tests et d'optimiser vos tests, consultez [Extension des environnements de test personnalisés dans Device Farm](#).

Ajouter des fichiers supplémentaires à votre package de test dans Device Farm

Vous souhaitez peut-être utiliser des fichiers supplémentaires dans le cadre de vos tests, soit en tant que fichiers de configuration supplémentaires, soit en tant que données de test supplémentaires. Vous pouvez ajouter ces fichiers supplémentaires à votre package de test avant de le télécharger AWS Device Farm, puis y accéder depuis le mode environnement personnalisé. Fondamentalement, tous les formats de téléchargement de packages de test (ZIP, IPA, APK, JAR, etc.) sont des formats d'archive de packages qui prennent en charge les opérations ZIP standard.

Vous pouvez ajouter des fichiers à votre archive de test avant de la télécharger à l'aide AWS Device Farm de la commande suivante :

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

Pour un répertoire de fichiers supplémentaires :

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

Ces commandes fonctionnent comme prévu pour tous les formats de téléchargement de packages de test, à l'exception des fichiers IPA. Pour les fichiers IPA, en particulier lorsqu'ils sont utilisés avec XCUITests, nous vous recommandons de placer les fichiers supplémentaires dans un emplacement légèrement différent en raison de la manière dont les packages de test AWS Device Farm iOS sont résignés. Lors de la création de votre test iOS, le répertoire de l'application de test sera situé dans un autre répertoire nommé *Payload*.

Par exemple, voici à quoi peut ressembler un tel répertoire de test iOS :

```
$ tree
.
```



```

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

```

Pour ces XCUITest packages, ajoutez des fichiers supplémentaires dans le répertoire se terminant par « *.app* inside » du *Payload* répertoire. Par exemple, les commandes suivantes indiquent comment ajouter un fichier à ce package de test :

```

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

```

Lorsque vous ajoutez un fichier à votre package de test, vous pouvez vous attendre à un comportement d'interaction légèrement différent en AWS Device Farm fonction de son format de téléchargement. Si le téléchargement a utilisé l'extension de fichier ZIP, le téléchargement AWS Device Farm sera automatiquement décompressé avant le test et les fichiers décompressés resteront à l'emplacement où se trouve la variable d'environnement. `$DEVICEFARM_TEST_PACKAGE_PATH` (Cela signifie que si vous ajoutiez un fichier appelé `extra_file` à la racine de l'archive comme dans le premier exemple, il sera localisé `$DEVICEFARM_TEST_PACKAGE_PATH/extra_file` pendant le test).

Pour utiliser un exemple plus pratique, si vous êtes un utilisateur d'Appium TestNG qui souhaite inclure un `testng.xml` fichier dans votre test, vous pouvez l'inclure dans votre archive à l'aide de la commande suivante :

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

Vous pouvez ensuite modifier votre commande de test dans le mode d'environnement personnalisé comme suit :

```
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
```

Si l'extension de téléchargement de votre package de test n'est pas ZIP (par exemple, un fichier APK, IPA ou JAR), le fichier de package téléchargé se trouve à l'`$DEVICEFARM_TEST_PACKAGE_PATH` adresse. Comme il s'agit toujours de fichiers au format archive, vous pouvez décompresser le fichier afin d'accéder aux fichiers supplémentaires de l'intérieur. Par exemple, la commande suivante décompressera le contenu du package de test (pour les fichiers APK, IPA ou JAR) dans le `/tmp` répertoire :

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

Dans le cas d'un fichier APK ou JAR, vous trouverez vos fichiers supplémentaires décompressés `/tmp` dans le répertoire (par exemple, `/tmp/extra_file`). Dans le cas d'un fichier IPA, comme expliqué précédemment, les fichiers supplémentaires se trouveraient à un emplacement légèrement différent dans le dossier se terminant par `.app`, qui se trouve à l'intérieur du `Payload` répertoire. Par exemple, sur la base de l'exemple IPA ci-dessus, le fichier se trouverait à l'emplacement `/tmp/Payload/ADFiOSReferenceAppUITests-Runner.app/extra_file` (référéncable en tant que). `/tmp/Payload/*.app/extra_file`

Pour découvrir d'autres moyens d'étendre votre suite de tests et d'optimiser vos tests, consultez [Extension des environnements de test personnalisés dans Device Farm](#).

Accès à distance dans AWS Device Farm

L'accès à distance vous permet de balayer, d'appuyer et d'interagir avec un appareil par le biais de votre navigateur Web en temps réel afin de tester les fonctionnalités et de reproduire les problèmes des clients. Vous interagissez avec un appareil spécifique en créant une session d'accès à distance avec celui-ci.

Une session dans Device Farm est une interaction en temps réel avec un appareil physique réel hébergé dans un navigateur Web. Une session affiche le seul appareil que vous sélectionnez lorsque vous démarrez la session. Un utilisateur peut démarrer plusieurs sessions à la fois, le nombre total d'appareils simultanés étant limité par le nombre d'emplacements d'appareils dont vous disposez. Vous pouvez acheter des emplacements d'appareils en fonction du type d'appareil (appareils Android ou iOS). Pour plus d'informations, consultez [Tarification de Device Farm](#).

Device Farm propose actuellement un sous-ensemble d'appareils pour les tests d'accès à distance. De nouveaux appareils sont constamment ajoutés à ce groupe d'appareils.

Device Farm capture une vidéo de chaque session d'accès à distance et génère des journaux d'activité pendant la session. Ces résultats incluent toutes les informations que vous fournissez au cours d'une session.

Note

Pour des raisons de sécurité, nous vous recommandons d'éviter de fournir ou de saisir des informations sensibles, telles que des numéros de compte, des informations de connexion personnelles et autres détails, pendant une session d'accès à distance. Si possible, utilisez des alternatives développées spécifiquement pour les tests, telles que des comptes de test.

Rubriques

- [Création d'une session d'accès à distance dans AWS Device Farm](#)
- [Utilisation d'une session d'accès à distance dans AWS Device Farm](#)
- [Récupération des résultats d'une session d'accès à distance dans AWS Device Farm](#)

Création d'une session d'accès à distance dans AWS Device Farm

Pour plus d'informations sur les sessions d'accès à distance, consultez [Séances](#).

- [Conditions préalables](#)
- [Création d'une session à distance](#)
- [Étapes suivantes](#)

Conditions préalables

- Créez un projet dans Device Farm. Suivez les instructions de [Création d'un projet dans AWS Device Farm](#), puis revenez à cette page.

Création d'une session à distance

Console

1. Connectez-vous à la console Device Farm à l'adresse <https://console.aws.amazon.com/devicefarm>.
2. Sur le panneau de navigation de Device Farm, choisissez Mobile Device Testing, puis Projects.
3. Si vous avez déjà un projet, sélectionnez-le dans la liste. Dans le cas contraire, créez un projet en suivant les instructions de [Création d'un projet dans AWS Device Farm](#).
4. Dans l'onglet Accès à distance, choisissez Créer une session d'accès à distance.
5. Choisissez un appareil pour votre session. Vous pouvez choisir parmi la liste des appareils disponibles ou rechercher un appareil à l'aide de la barre de recherche en haut de la liste.
6. (Facultatif) Incluez une application et des applications auxiliaires dans le cadre de la session. Il peut s'agir d'applications récemment téléchargées ou d'applications précédemment téléchargées dans le cadre de ce projet au cours des 30 derniers jours (après 30 jours, les téléchargements d'applications [expirent](#)).
7. Dans Session name (Nom de session), entrez le nom de la session.
8. Choisissez Confirm and start session (Confirmer et démarrer la session).

AWS CLI

Remarque : ces instructions concernent uniquement la création d'une session d'accès à distance. Pour obtenir des instructions sur la façon de télécharger une application à utiliser pendant votre session, consultez la section [Automatisation des téléchargements d'applications](#).

Tout d'abord, vérifiez que votre version de l'AWS CLI est up-to-date correcte en [téléchargeant et en installant la dernière version](#).

 Important

Certaines commandes mentionnées dans ce document ne sont pas disponibles dans les anciennes versions de l'AWS CLI.

Ensuite, vous pouvez déterminer sur quel appareil vous souhaitez effectuer le test :

```
$ aws devicefarm list-devices
```

Cela affichera des résultats tels que les suivants :

```
{
  "devices":
  [
    {
      "arn": "arn:aws:devicefarm:us-
west-2::device:DE5BD47FF3BD42C3A14BF7A6EFB1BFE7",
      "name": "Google Pixel 8",
      "remoteAccessEnabled": true,
      "availability": "HIGHLY_AVAILABLE"
      ...
    },
    ...
  ]
}
```

Vous pouvez ensuite créer votre session d'accès à distance avec l'ARN de l'appareil de votre choix :

```
$ aws devicefarm create-remote-access-session \
  --project-arn arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef \
  --device-arn arn:aws:devicefarm:us-west-2::device:DE5BD47FF3BD42C3A14BF7A6EFB1BFE7
\
  --app-arn arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
```

```
--configuration '{
  "auxiliaryApps": [
    "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
    "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
  ]
}'
```

Cela affichera des résultats tels que les suivants :

```
{
  "remoteAccessSession": {
    "arn": "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/000000",
    "name": "Google Pixel 8",
    "status": "PENDING",
    ...
  }
}
```

Maintenant, éventuellement, nous pouvons interroger et attendre que la session soit prête :

```
$ POLL_INTERVAL=3
TIMEOUT=600
DEADLINE=$(( $(date +%s) + TIMEOUT ))

while [[ "$(date +%s)" -lt "$DEADLINE" ]]; do

  STATUS=$(aws devicefarm get-remote-access-session \
    --arn "$DEVICE_FARM_SESSION_ARN" \
    --query 'remoteAccessSession.status' \
    --output text)

  case "$STATUS" in
    RUNNING)
      echo "Session is ready with status: $STATUS"
      break
      ;;
    STOPPING|COMPLETED)
      echo "Session ended early with status: $STATUS"
      exit 1
      ;;
  esac
done
```

esac**done**

Python

Remarque : ces instructions concernent uniquement la création d'une session d'accès à distance. Pour obtenir des instructions sur la façon de télécharger une application à utiliser pendant votre session, consultez la section [Automatisation des téléchargements d'applications](#).

Cet exemple trouve d'abord n'importe quel appareil Google Pixel disponible sur Device Farm, puis crée une session d'accès à distance avec celui-ci et attend que la session soit en cours.

```
import random
import time
import boto3

client = boto3.client("devicefarm", region_name="us-west-2")

# 1) Gather all matching devices via paginated ListDevices with filters
filters = [
    {"attribute": "MODEL", "operator": "CONTAINS", "values": ["Pixel"]},
    {"attribute": "AVAILABILITY", "operator": "EQUALS", "values": ["AVAILABLE"]},
]

matching_arns = []
next_token = None
while True:
    args = {"filters": filters}
    if next_token:
        args["nextToken"] = next_token
    page = client.list_devices(**args)
    for d in page.get("devices", []):
        matching_arns.append(d["arn"])
    next_token = page.get("nextToken")
    if not next_token:
        break

if not matching_arns:
    raise RuntimeError("No available Google Pixel device found.")

# Randomly select one device from the full matching set
device_arn = random.choice(matching_arns)
```



```
print("Selected device ARN:", device_arn)

# 2) Create remote access session and wait until RUNNING
resp = client.create_remote_access_session(
    projectArn="arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef",
    deviceArn=device_arn,
    appArn="arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
    # optional
    configuration={
        "auxiliaryApps": [ # optional
            "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
            "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
        ]
    },
)

session_arn = resp["remoteAccessSession"]["arn"]
print(f"Created Remote Access Session: {session_arn}")

poll_interval = 3
timeout = 600
deadline = time.time() + timeout
terminal_states = ["STOPPING", "COMPLETED"]

while True:
    out = client.get_remote_access_session(arn=session_arn)
    status = out["remoteAccessSession"]["status"]
    print(f"Current status: {status}")

    if status == "RUNNING":
        print(f"Session is ready with status: {status}")
        break
    if status in terminal_states:
        raise RuntimeError(f"Session ended early with status: {status}")
    if time.time() >= deadline:
        raise RuntimeError("Timed out waiting for session to be ready.")
    time.sleep(poll_interval)
```

Java

Remarque : ces instructions concernent uniquement la création d'une session d'accès à distance. Pour obtenir des instructions sur la façon de télécharger une application à utiliser pendant votre session, consultez la section [Automatisation des téléchargements d'applications](#).

Remarque : cet exemple utilise le SDK AWS pour Java v2 et est compatible avec les versions 11 et supérieures du JDK.

Cet exemple trouve d'abord n'importe quel appareil Google Pixel disponible sur Device Farm, puis crée une session d'accès à distance avec celui-ci et attend que la session soit en cours.

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.concurrent.ThreadLocalRandom;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import
    software.amazon.awssdk.services.devicefarm.model.CreateRemoteAccessSessionConfiguration;
import
    software.amazon.awssdk.services.devicefarm.model.CreateRemoteAccessSessionRequest;
import
    software.amazon.awssdk.services.devicefarm.model.CreateRemoteAccessSessionResponse;
import software.amazon.awssdk.services.devicefarm.model.Device;
import software.amazon.awssdk.services.devicefarm.model.DeviceFilter;
import software.amazon.awssdk.services.devicefarm.model.DeviceFilterAttribute;
import
    software.amazon.awssdk.services.devicefarm.model.GetRemoteAccessSessionRequest;
import
    software.amazon.awssdk.services.devicefarm.model.GetRemoteAccessSessionResponse;
import software.amazon.awssdk.services.devicefarm.model.ListDevicesRequest;
import software.amazon.awssdk.services.devicefarm.model.ListDevicesResponse;
import software.amazon.awssdk.services.devicefarm.model.RuleOperator;

public class CreateRemoteAccessSession {
    public static void main(String[] args) throws Exception {
        DeviceFarmClient client = DeviceFarmClient.builder()
            .region(Region.US_WEST_2)
            .build();

        String projectArn = "arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef";
```

```

String appArn      = "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
String aux1        = "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
String aux2        = "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789

// 1) Gather all matching devices via paginated ListDevices with filters
List<DeviceFilter> filters = Arrays.asList(
    DeviceFilter.builder()
        .attribute(DeviceFilterAttribute.MODEL)
        .operator(RuleOperator.CONTAINS)
        .values("Pixel")
        .build(),
    DeviceFilter.builder()
        .attribute(DeviceFilterAttribute.AVAILABILITY)
        .operator(RuleOperator.EQUALS)
        .values("AVAILABLE")
        .build()
);

List<String> matchingDeviceArns = new ArrayList<>();
String next = null;
do {
    ListDevicesResponse page = client.listDevices(
        ListDevicesRequest.builder().filters(filters).nextToken(next).build());
    for (Device d : page.devices()) {
        matchingDeviceArns.add(d.arn());
    }
    next = page.nextToken();
} while (next != null);

if (matchingDeviceArns.isEmpty()) {
    throw new RuntimeException("No available Google Pixel device found.");
}

// Randomly select one device from the full matching set
String deviceArn = matchingDeviceArns.get(
    ThreadLocalRandom.current().nextInt(matchingDeviceArns.size()));
System.out.println("Selected device ARN: " + deviceArn);

// 2) Create Remote Access session and wait until it is RUNNING
CreateRemoteAccessSessionConfiguration cfg =
CreateRemoteAccessSessionConfiguration.builder()

```

```
.auxiliaryApps(Arrays.asList(aux1, aux2))
    .build();

CreateRemoteAccessSessionResponse res = client.createRemoteAccessSession(
    CreateRemoteAccessSessionRequest.builder()
        .projectArn(projectArn)
        .deviceArn(deviceArn)
        .appArn(appArn)           // optional
        .configuration(cfg)       // optional
        .build());

String sessionArn = res.remoteAccessSession().arn();
System.out.println("Created Remote Access Session: " + sessionArn);

int pollIntervalMs = 3000;
long timeoutMs = 600_000L;
long deadline = System.currentTimeMillis() + timeoutMs;

while (true) {
    GetRemoteAccessSessionResponse get = client.getRemoteAccessSession(
        GetRemoteAccessSessionRequest.builder().arn(sessionArn).build());
    String status = get.remoteAccessSession().statusAsString();
    System.out.println("Current status: " + status);

    if ("RUNNING".equals(status)) {
        System.out.println("Session is ready with status: " + status);
        break;
    }
    if ("STOPPING".equals(status) || "COMPLETED".equals(status)) {
        throw new RuntimeException("Session ended early with status: " + status);
    }
    if (System.currentTimeMillis() >= deadline) {
        throw new RuntimeException("Timed out waiting for session to be ready.");
    }
    Thread.sleep(pollIntervalMs);
}
}
```

JavaScript

Remarque : ces instructions concernent uniquement la création d'une session d'accès à distance. Pour obtenir des instructions sur la façon de télécharger une application à utiliser pendant votre session, consultez la section [Automatisation des téléchargements d'applications](#).

Remarque : cet exemple utilise le SDK AWS pour la JavaScript version 3.

Cet exemple trouve d'abord n'importe quel appareil Google Pixel disponible sur Device Farm, puis crée une session d'accès à distance avec celui-ci et attend que la session soit en cours.

```
import {
  DeviceFarmClient,
  ListDevicesCommand,
  CreateRemoteAccessSessionCommand,
  GetRemoteAccessSessionCommand,
} from "@aws-sdk/client-device-farm";

const client = new DeviceFarmClient({ region: "us-west-2" });

// 1) Gather all matching devices via paginated ListDevices with filters
const filters = [
  { attribute: "MODEL", operator: "CONTAINS", values: ["Pixel"] },
  { attribute: "AVAILABILITY", operator: "EQUALS", values: ["AVAILABLE"] },
];

let nextToken;
const matching = [];

while (true) {
  const page = await client.send(new ListDevicesCommand({ filters, nextToken }));
  for (const d of page.devices ?? []) {
    matching.push(d.arn);
  }
  nextToken = page.nextToken;
  if (!nextToken) break;
}

if (matching.length === 0) {
  throw new Error("No available Google Pixel device found.");
}

// Randomly select one device from the full matching set
```

```
const deviceArn = matching[Math.floor(Math.random() * matching.length)];
console.log("Selected device ARN:", deviceArn);

// 2) Create remote access session and wait until RUNNING
const out = await client.send(new CreateRemoteAccessSessionCommand({
  projectArn: "arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef",
  deviceArn,
  appArn: "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
optional
  configuration: {
    auxiliaryApps: [ // optional
      "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
      "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
    ],
  },
}));

const sessionArn = out.remoteAccessSession?.arn;
console.log("Created Remote Access Session:", sessionArn);

const pollIntervalMs = 3000;
const timeoutMs = 600000;
const deadline = Date.now() + timeoutMs;

while (true) {
  const get = await client.send(new GetRemoteAccessSessionCommand({ arn:
sessionArn }));
  const status = get.remoteAccessSession?.status;
  console.log("Current status:", status);

  if (status === "RUNNING") {
    console.log("Session is ready with status:", status);
    break;
  }
  if (status === "STOPPING" || status === "COMPLETED") {
    throw new Error(`Session ended early with status: ${status}`);
  }
  if (Date.now() >= deadline) {
    throw new Error("Timed out waiting for session to be ready.");
  }
}
```

```
    await new Promise((r) => setTimeout(r, pollIntervalMs));  
}
```

C#

Remarque : ces instructions concernent uniquement la création d'une session d'accès à distance. Pour obtenir des instructions sur la façon de télécharger une application à utiliser pendant votre session, consultez la section [Automatisation des téléchargements d'applications](#).

Cet exemple trouve d'abord n'importe quel appareil Google Pixel disponible sur Device Farm, puis crée une session d'accès à distance avec celui-ci et attend que la session soit en cours.

```
using System;  
using System.Collections.Generic;  
using System.Threading.Tasks;  
using Amazon;  
using Amazon.DeviceFarm;  
using Amazon.DeviceFarm.Model;  
  
class Program  
{  
    static async Task Main()  
    {  
        var client = new AmazonDeviceFarmClient(RegionEndpoint.USWest2);  
  
        // 1) Gather all matching devices via paginated ListDevices with filters  
        var filters = new List<DeviceFilter>  
        {  
            new DeviceFilter { Attribute = DeviceAttribute.MODEL, Operator =  
                RuleOperator.CONTAINS, Values = new List<string>{ "Pixel" } },  
            new DeviceFilter { Attribute = DeviceAttribute.AVAILABILITY, Operator =  
                RuleOperator.EQUALS, Values = new List<string>{ "AVAILABLE" } },  
        };  
  
        var matchingArns = new List<string>();  
        string nextToken = null;  
  
        do  
        {  
            var list = await client.ListDevicesAsync(new ListDevicesRequest  
            {  
                Filters = filters,  
                NextToken = nextToken  
            });  
            matchingArns.AddRange(list.Devices.Select(d => d.Arn));  
            nextToken = list.NextToken;  
        } while (list.NextToken != null);  
    }  
}
```

```

    });

    foreach (var d in list.Devices)
        matchingArns.Add(d.Arn);

    nextToken = list.NextToken;
}
while (nextToken != null);

if (matchingArns.Count == 0)
    throw new Exception("No available Google Pixel device found.");

// Randomly select one device from the full matching set
var rnd = new Random();
var deviceArn = matchingArns[rnd.Next(matchingArns.Count)];
Console.WriteLine($"Selected device ARN: {deviceArn}");

// 2) Create remote access session and wait until RUNNING
var request = new CreateRemoteAccessSessionRequest
{
    ProjectArn = "arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef",
    DeviceArn = deviceArn,
    AppArn = "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
optional
    Configuration = new CreateRemoteAccessSessionConfiguration
    {
        AuxiliaryApps = new List<string>
        {
            "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
            "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
        }
    }
};

request.Configuration.AuxiliaryApps.RemoveAll(string.IsNullOrEmpty);

var response = await client.CreateRemoteAccessSessionAsync(request);
var sessionArn = response.RemoteAccessSession.Arn;
Console.WriteLine($"Created Remote Access Session: {sessionArn}");

```



```

var pollIntervalMs = 3000;
var timeoutMs = 600000;
var deadline = DateTime.UtcNow.AddMilliseconds(timeoutMs);

while (true)
{
    var get = await client.GetRemoteAccessSessionAsync(new
GetRemoteAccessSessionRequest { Arn = sessionArn });
    var status = get.RemoteAccessSession.Status.Value;
    Console.WriteLine($"Current status: {status}");

    if (status == "RUNNING")
    {
        Console.WriteLine($"Session is ready with status: {status}");
        break;
    }
    if (status == "STOPPING" || status == "COMPLETED")
    {
        throw new Exception($"Session ended early with status: {status}");
    }
    if (DateTime.UtcNow >= deadline)
    {
        throw new TimeoutException("Timed out waiting for session to be
ready.");
    }

    await Task.Delay(pollIntervalMs);
}
}
}

```

Ruby

Remarque : ces instructions concernent uniquement la création d'une session d'accès à distance. Pour obtenir des instructions sur la façon de télécharger une application à utiliser pendant votre session, consultez la section [Automatisation des téléchargements d'applications](#).

Cet exemple trouve d'abord n'importe quel appareil Google Pixel disponible sur Device Farm, puis crée une session d'accès à distance avec celui-ci et attend que la session soit en cours.

```

require 'aws-sdk-devicefarm'

client = Aws::DeviceFarm::Client.new(region: 'us-west-2')

```

```

# 1) Gather all matching devices via paginated ListDevices with filters
filters = [
  { attribute: 'MODEL',          operator: 'CONTAINS', values: ['Pixel'] },
  { attribute: 'AVAILABILITY', operator: 'EQUALS',   values: ['AVAILABLE'] },
]

matching_arns = []
next_token = nil
loop do
  resp = client.list_devices(filters: filters, next_token: next_token)
  resp.devices.each { |d| matching_arns << d.arn }
  next_token = resp.next_token
  break unless next_token
end

abort "No available Google Pixel device found." if matching_arns.empty?

# Randomly select one device from the full matching set
device_arn = matching_arns.sample
puts "Selected device ARN: #{device_arn}"

# 2) Create remote access session and wait until RUNNING
resp = client.create_remote_access_session(
  project_arn: "arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef",
  device_arn:  device_arn,
  app_arn:     "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
# optional
  configuration: {
    auxiliary_apps: [ # optional
      "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
      "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
    ].compact
  }
)

session_arn = resp.remote_access_session.arn
puts "Created Remote Access Session: #{session_arn}"

poll_interval = 3

```

```
timeout = 600
deadline = Time.now + timeout
terminal = %w[STOPPING COMPLETED]

loop do
  get = client.get_remote_access_session(arn: session_arn)
  status = get.remote_access_session.status
  puts "Current status: #{status}"

  if status == 'RUNNING'
    puts "Session is ready with status: #{status}"
    break
  end

  abort "Session ended early with status: #{status}" if terminal.include?(status)
  abort "Timed out waiting for session to be ready." if Time.now >= deadline
  sleep poll_interval
end
```

Étapes suivantes

Device Farm démarre la session dès que l'appareil et l'infrastructure demandés sont disponibles, généralement en quelques minutes. La boîte de dialogue Device Requested s'affiche jusqu'au démarrage de la session. Pour annuler la demande de session, choisissez Cancel request (Annuler la demande).

Si l'appareil sélectionné est indisponible ou occupé, l'état de la session indique « Appareil en attente », ce qui indique que vous devrez peut-être attendre un certain temps avant que l'appareil ne soit disponible pour le test.

Si votre compte a atteint sa limite de simultanéité pour les appareils publics avec ou sans compteur, le statut de la session indique « En attente de simultanéité ». Pour les emplacements illimités, vous pouvez augmenter la simultanéité en [achetant davantage d'emplacements](#) pour appareils. Pour les pay-as-you-go appareils équipés de compteurs, contactez AWS via un ticket d'assistance pour demander [une augmentation du quota de service](#).

Lorsque la configuration de la session commence, elle affiche d'abord le statut En cours, puis le statut Connexion pendant que votre navigateur Web local tente d'ouvrir une connexion à distance avec l'appareil.

Après le démarrage d'une session, si vous fermez le navigateur ou l'onglet du navigateur sans arrêter la session, ou si la connexion entre le navigateur et Internet est perdue, la session reste active pendant cinq minutes. Device Farm met ensuite fin à la session. Votre compte est facturé pour le temps d'inactivité.

Une fois la session démarrée, vous pouvez interagir avec l'appareil dans le navigateur Web ou tester l'appareil à l'aide d'[Appium](#).

Utilisation d'une session d'accès à distance dans AWS Device Farm

Pour de plus amples informations sur l'exécution des tests interactifs sur des applications Android et iOS grâce à des sessions d'accès à distance, veuillez consulter [Séances](#).

- [Conditions préalables](#)
- [Utiliser une session dans la console Device Farm](#)
- [Étapes suivantes](#)
- [Trucs et astuces](#)

Conditions préalables

- Créez une session. Suivez les instructions de [Création d'une session](#), puis revenez à cette page.

Utiliser une session dans la console Device Farm

Dès que l'appareil que vous avez demandé pour une session d'accès à distance devient disponible, la console affiche son écran. La durée maximale de la session est de 150 minutes. Le temps restant dans la session apparaît dans le champ Temps restant à côté du nom de l'appareil.

Installation d'une application

Pour installer une application sur l'appareil de session, dans Installer des applications, sélectionnez Choisir un fichier, puis choisissez le fichier .apk (Android) ou le fichier .ipa (iOS) que vous souhaitez installer. Les applications que vous exécutez lors d'une session d'accès à distance ne nécessitent aucune instrumentation de test, ni aucune mise en service.

 Note

Lorsque vous chargez une application, l'application devient parfois disponible après un certain délai. Un message de confirmation s'affichera pour vous indiquer si l'application a été correctement installée ou non.

Contrôle de l'appareil

Vous pouvez interagir avec l'appareil affiché dans la console comme vous le feriez avec l'appareil physique réel, en utilisant votre souris ou tout autre appareil équivalent pour la fonction tactile et le clavier à l'écran de l'appareil. Pour les appareils Android, il y a des boutons View controls (Commandes d'affichage) qui fonctionnent de la même manière que les boutons Accueil et Retour sur un appareil Android. Pour les appareils iOS, il existe un bouton Home (Accueil) qui fonctionne comme le bouton d'accueil sur un appareil iOS. Vous pouvez également basculer entre les applications exécutées sur l'appareil en choisissant Recent Apps.

Basculer entre le mode portrait et le mode paysage

Vous pouvez également passer du mode portrait (vertical) au mode paysage (horizontal) pour les appareils que vous utilisez.

Étapes suivantes

Device Farm poursuit la session jusqu'à ce que vous l'arrêtiez manuellement ou que le délai de 150 minutes soit atteint. Pour terminer la session, choisissez Arrêter la session. Lorsque la session s'arrête, vous pouvez accéder à la vidéo qui a été capturée et aux journaux qui ont été générés. Pour de plus amples informations, veuillez consulter [Récupération des résultats de session](#).

Trucs et astuces

Vous pouvez rencontrer des problèmes de performances lors de la session d'accès à distance dans certaines AWS régions. Cela est dû en partie à la latence dans certaines régions. Si vous rencontrez des problèmes de performances, laissez à la session d'accès à distance la possibilité de rattraper son retard avant d'interagir à nouveau avec l'application.

Récupération des résultats d'une session d'accès à distance dans AWS Device Farm

Pour plus d'informations sur les sessions, consultez [Séances](#).

- [Conditions préalables](#)
- [Afficher les détails de la session](#)
- [Téléchargement de vidéos ou de journaux de session](#)

Conditions préalables

- Terminez une session. Suivez les instructions de [Utilisation d'une session d'accès à distance dans AWS Device Farm](#), puis revenez à cette page.

Afficher les détails de la session

Lorsqu'une session d'accès à distance se termine, la console Device Farm affiche un tableau contenant des détails sur l'activité pendant la session. Pour plus d'informations, consultez la section relative à l'[analyse des informations de journal](#).

Pour revenir ultérieurement aux informations d'une session :

1. Sur le panneau de navigation de Device Farm, choisissez Mobile Device Testing, puis Projects.
2. Choisissez le projet contenant la session.
3. Choisissez Accès à distance, puis sélectionnez la session que vous souhaitez consulter dans la liste.

Téléchargement de vidéos ou de journaux de session

Lorsqu'une session d'accès à distance se termine, la console Device Farm permet d'accéder à une capture vidéo de la session et aux journaux d'activité. Dans les résultats de la session, choisissez l'onglet Files (Fichiers) pour obtenir une liste de liens vers les vidéos et les journaux de la session. Vous pouvez consulter ces fichiers dans le navigateur ou les enregistrer localement.

Tests d'Appium dans AWS Device Farm

Au cours d'une session d'accès à distance, vous pouvez exécuter des tests Appium depuis votre environnement local, en ciblant l'appareil de la session à l'aide d'un point de terminaison Appium géré. Avec un point de terminaison Appium, vous pouvez développer, tester et exécuter du code Appium avec un feedback rapide et une itération rapide. Cette approche des tests côté client offre la flexibilité de se connecter à un appareil Device Farm depuis n'importe quel environnement client Appium de votre choix.

Pour compléter les tests côté client, Device Farm prend également en charge l'exécution de tests sur l'infrastructure gérée par le service, appelés exécution côté serveur. [Dans cette approche, vous pouvez télécharger votre application et vos tests sur le service, puis exécuter les tests en parallèle sur plusieurs appareils à l'aide d'hôtes de test gérés par le service.](#) Cette approche s'adapte bien aux tests indépendants sur de nombreux appareils, ainsi qu'aux tests dans le contexte d'un CI/CD pipeline.

Pour en savoir plus sur l'exécution côté serveur, consultez. [Cadres de test et tests intégrés](#)

Rubriques

- [Qu'est-ce qu'un terminal Appium ?](#)
- [Commencer à tester Appium](#)
- [Interaction avec l'appareil à l'aide d'Appium](#)
- [Consulter les journaux de votre serveur Appium](#)
- [Fonctionnalités et commandes Appium prises en charge](#)

Qu'est-ce qu'un terminal Appium ?

[Appium](#) est un framework de test de logiciels open source populaire permettant de tester des applications Web natives, hybrides et mobiles sur différents appareils, y compris les téléphones mobiles et les tablettes, pour iOS et Android. Il permet aux développeurs et aux ingénieurs QA (assurance qualité) d'écrire des scripts capables de contrôler à distance un appareil, de simuler les interactions avec les utilisateurs et de vérifier que l'application testée se comporte comme prévu. Appium interagit avec les applications du point de vue de l'utilisateur final, ce qui permet aux testeurs de développer des tests simulant la manière dont les utilisateurs réels utiliseront l'application pour leurs tests.

Appium est basé sur le modèle client-serveur, dans lequel un client local demande à un serveur Appium (local ou distant) de commander un appareil en son nom. Le serveur Appium gère un pilote pour communiquer avec l'appareil, tel que le [UIAutomator2 pilote](#) pour Android ou le [XCUITest pilote](#) pour iOS. Toutes les commandes sont WebDriver conformes aux normes du [W3C](#) relatives au contrôle d'un appareil.

Le point de terminaison Appium de Device Farm expose l'URL du serveur Appium pour l'appareil dans votre session d'accès à distance. L'URL du point de terminaison Appium sera spécifique à cet appareil dans cette session et restera valide pendant toute la durée de la session, ce qui vous permettra d'itérer sur le même appareil sans temps de configuration supplémentaire. Pour plus d'informations sur l'accès à distance, veuillez consulter [Accès à distance](#).

Commencer à tester Appium

Pour la plupart des utilisateurs d'Appium, l'utilisation de Device Farm pour les tests Appium ne nécessite que des modifications mineures de votre configuration de test existante.

De manière générale, l'utilisation de Device Farm pour les tests Appium côté client comporte trois étapes :

1. Tout d'abord, vous devez [créer une session d'accès à distance](#) pour tester un appareil Device Farm. Vous pouvez inclure vos applications dans le cadre de votre demande d'accès à distance ou installer des applications après le début de la session.
2. Une fois la session en cours d'exécution, vous pouvez [copier l'URL du point de terminaison Appium](#) et l'utiliser soit via un outil autonome (comme Appium [Inspector](#)), soit à partir de votre code de test Appium dans votre IDE. L'URL sera valide pendant toute la durée de la session d'accès à distance.
3. Enfin, une fois que votre test Appium a commencé, vous pouvez [consulter les journaux de votre serveur Appium](#) en direct pendant l'exécution du test, parallèlement au flux vidéo de votre appareil.

Interaction avec l'appareil à l'aide d'Appium

Une fois que vous avez [créé une session d'accès à distance](#), l'appareil sera disponible pour les tests Appium. Pendant toute la durée de la session d'accès à distance, vous pouvez exécuter autant de sessions Appium que vous le souhaitez sur l'appareil, sans aucune limite quant aux clients que vous utilisez. Par exemple, vous pouvez commencer par exécuter un test en utilisant votre code Appium local à partir de votre IDE, puis passer à l'utilisation d'Appium Inspector pour résoudre les

problèmes que vous rencontrez. La session peut durer jusqu'à [150 minutes](#), mais s'il n'y a aucune activité pendant plus de 5 minutes (via la console interactive ou via le point de terminaison Appium), la session expirera.

Utilisation d'applications pour les tests avec votre session Appium

Device Farm vous permet d'utiliser vos applications dans le cadre de votre demande de création de session d'accès à distance ou d'installer des applications pendant la session d'accès à distance elle-même. Ces applications sont automatiquement installées sur l'appareil testé et sont injectées en tant que fonctionnalités par défaut pour toute demande de session Appium. Lorsque vous créez une session d'accès à distance, vous avez la possibilité de transmettre un ARN d'application, qui sera utilisé par défaut comme `appium:app` fonctionnalité pour toutes les sessions Appium suivantes, ainsi qu'une application auxiliaire ARNs, qui sera utilisée comme fonctionnalité. `appium:otherApps`

Par exemple, si vous créez une session d'accès à distance en utilisant une application `com.aws.devicefarm.sample` comme application et comme application auxiliaire, lorsque vous créez une session Appium, elle disposera de fonctionnalités similaires aux suivantes :

```
{
  "value":
  {
    "sessionId": "abcdef123456-1234-5678-abcd-abcdef123456",
    "capabilities":
    {
      "app": "/tmp/com.aws.devicefarm.sample.apk",
      "otherApps": "[\"/tmp/com.aws.devicefarm.other.sample.apk\"]",
      ...
    }
  }
}
```

Au cours de votre session, vous pouvez installer des applications supplémentaires (soit dans la console, soit à l'aide de l'[InstallToRemoteAccessSessionAPI](#)). Elles remplaceront toutes les applications existantes précédemment utilisées comme `appium:app` fonctionnalité. Si les applications précédemment utilisées ont un nom de package distinct, elles resteront sur l'appareil et seront utilisées dans le cadre de la `appium:otherApps` fonctionnalité.

Par exemple, si vous utilisez initialement une application `com.aws.devicefarm.sample` lors de la création de votre session d'accès à distance, mais que vous installez ensuite une nouvelle application

nommée `com.aws.devicefarm.other.sample` pendant la session, vos sessions Appium auront des fonctionnalités similaires aux suivantes :

```
{
  "value":
  {
    "sessionId": "abcdef123456-1234-5678-abcd-abcdef123456",
    "capabilities":
    {
      "app": "/tmp/com.aws.devicefarm.other.sample.apk",
      "otherApps": "[\"/tmp/com.aws.devicefarm.sample.apk\"]",
      ...
    }
  }
}
```

Si vous préférez, vous pouvez spécifier explicitement les fonctionnalités de votre application en utilisant le nom de l'application (en utilisant les `appium:bundleId` fonctionnalités `appium:appPackage` ou pour Android et iOS respectivement).

Si vous testez une application Web, spécifiez la `browserName` capacité de votre demande de création de session Appium. Le Chrome navigateur est disponible sur tous les appareils Android et le Safari navigateur est disponible sur tous les appareils iOS.

Device Farm ne prend pas en charge la transmission d'une URL distante ou d'un chemin de système de fichiers local au `appium:app` cours d'une session d'accès à distance. Téléchargez des applications sur Device Farm et incluez-les plutôt dans la session.

Note

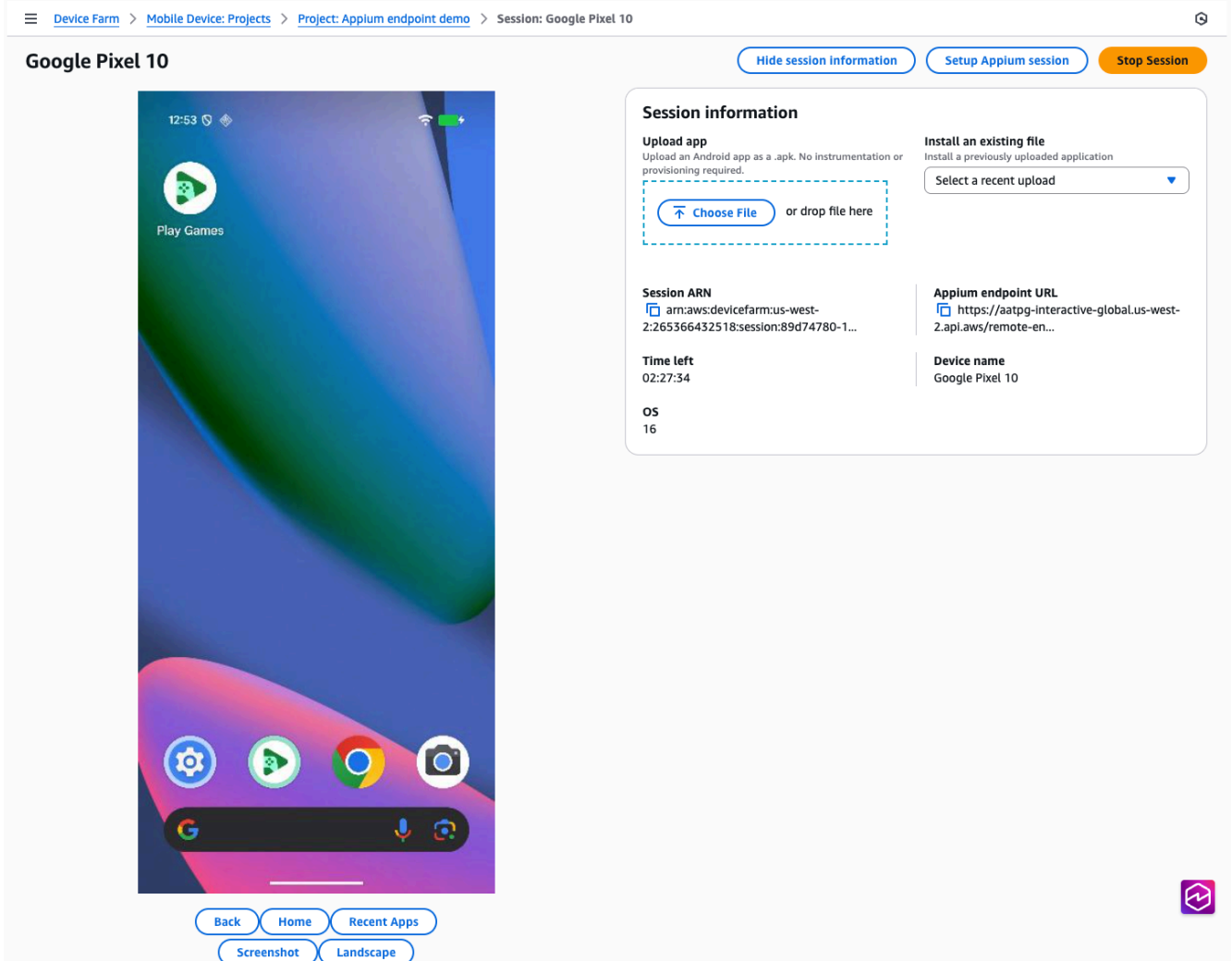
Pour plus d'informations sur le téléchargement automatique d'applications dans le cadre de votre session d'accès à distance, consultez la section [Automatisation des téléchargements d'applications](#).

Comment utiliser le point de terminaison Appium

Voici les étapes pour accéder au point de terminaison Appium de la session depuis la console, le AWS CLI, et le. AWS SDKs Ces étapes incluent comment commencer à exécuter des tests à l'aide de divers frameworks de test client Appium :

Console

1. Ouvrez la page de votre session d'accès à distance dans votre navigateur Web :



2. Pour exécuter une session à l'aide d'Appium Inspector, procédez comme suit :
 - a. Cliquez sur le bouton Configurer la session Appium
 - b. Suivez les instructions de la page pour savoir comment démarrer une session à l'aide d'Appium Inspector.
3. Pour exécuter un test Appium depuis votre IDE local, procédez comme suit :
 - a. Cliquez sur l'icône « copier » à côté du texte URL du point de terminaison Appium
 - b. Collez cette URL dans votre code Appium local là où vous spécifiez actuellement votre adresse distante ou votre exécuteur de commande. Pour des exemples

spécifiques à une langue, veuillez cliquer sur l'un des onglets de cette fenêtre d'exemple correspondant à la langue de votre choix.

AWS CLI

Tout d'abord, vérifiez que votre version de l'AWS CLI est up-to-date correcte en [téléchargeant et en installant la dernière version](#).

Important

Le champ de point de terminaison Appium n'est pas disponible dans les anciennes versions de l'AWS CLI.

Une fois votre session ouverte, l'URL du point de terminaison Appium sera disponible via un champ nommé `remoteDriverEndpoint` dans la réponse à un appel à l'[GetRemoteAccessSessionAPI](#) :

```
$ aws devicefarm get-remote-access-session \
  --arn "arn:aws:devicefarm:us-west-2:123456789876:session:abcdef123456-1234-5678-
  abcd-abcdef123456/abcdef123456-1234-5678-abcd-abcdef123456/000000"
```

Cela affichera des résultats tels que les suivants :

```
{
  "remoteAccessSession": {
    "arn": "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/000000",
    "name": "Google Pixel 8",
    "status": "RUNNING",
    "endpoints": {
      "remoteDriverEndpoint": "https://devicefarm-interactive-global.us-
west-2.api.aws/remote-endpoint/ABCD1234...",
      ...
    }
  }
}
```

Vous pouvez utiliser cette URL dans votre code Appium local partout où vous spécifiez actuellement votre adresse distante ou votre exécuteur de commande. Pour des exemples

spécifiques à une langue, veuillez cliquer sur l'un des onglets de cette fenêtre d'exemple correspondant à la langue de votre choix.

Pour un exemple d'interaction avec le point de terminaison directement depuis la ligne de commande, vous pouvez utiliser l'[outil de ligne de commande curl](#) pour appeler directement un WebDriver point de terminaison :

```
$ curl "https://devicefarm-interactive-global.us-west-2.api.aws/remote-endpoint/ABCD1234.../status"
```

Cela affichera des résultats tels que les suivants :

```
{
  "value":
  {
    "ready": true,
    "message": "The server is ready to accept new connections",
    "build":
    {
      "version": "2.5.1"
    }
  }
}
```

Python

Une fois votre session ouverte, l'URL du point de terminaison Appium sera disponible via un champ nommé `remoteDriverEndpoint` dans la réponse à un appel à l'[GetRemoteAccessSessionAPI](#) :

```
# To get the URL
import sys
import boto3
from botocore.exceptions import ClientError

def get_appium_endpoint() -> str:
    session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/000000"
    device_farm_client = boto3.client("devicefarm", region_name="us-west-2")

    try:
```

```

        resp = device_farm_client.get_remote_access_session(arn=session_arn)
    except ClientError as exc:
        sys.exit(f"Failed to call Device Farm: {exc}")

    remote_access_session = resp.get("remoteAccessSession", {})
    endpoints = remote_access_session.get("endpoints", {})
    endpoint = endpoints.get("remoteDriverEndpoint")

    if not endpoint:
        sys.exit("Device Farm response did not include
endpoints.remoteDriverEndpoint")

    return endpoint

# To use the URL
from appium import webdriver
from appium.options.android import UiAutomator2Options

opts = UiAutomator2Options()
driver = webdriver.Remote(get_appium_endpoint(), options=opts)
# ...
driver.quit()

```

Java

Remarque : cet exemple utilise le AWS SDK pour Java v2 et est compatible avec les versions 11 et supérieures du JDK.

Une fois votre session ouverte, l'URL du point de terminaison Appium sera disponible via un champ nommé `remoteDriverEndpoint` dans la réponse à un appel à l'[GetRemoteAccessSession](#) API :

```

// To get the URL
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import
    software.amazon.awssdk.services.devicefarm.model.GetRemoteAccessSessionRequest;
import
    software.amazon.awssdk.services.devicefarm.model.GetRemoteAccessSessionResponse;

public class AppiumEndpointBuilder {
    public static String getAppiumEndpoint() throws Exception {

```

```

        String session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/000000";

        try (DeviceFarmClient client = DeviceFarmClient.builder()
                .region(Region.US_WEST_2)
                .credentialsProvider(DefaultCredentialsProvider.create())
                .build()) {

            GetRemoteAccessSessionResponse resp = client.getRemoteAccessSession(
                GetRemoteAccessSessionRequest.builder().arn(session_arn).build()
            );

            String endpoint =
resp.remoteAccessSession().endpoints().remoteDriverEndpoint();
            if (endpoint == null || endpoint.isEmpty()) {
                throw new IllegalStateException("remoteDriverEndpoint missing from
response");
            }
            return endpoint;
        }
    }
}

// To use the URL
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.options.UiAutomator2Options;

import java.net.URL;

public class ExampleTest {
    public static void main(String[] args) throws Exception {
        String endpoint = AppiumEndpointBuilder.getAppiumEndpoint();
        UiAutomator2Options options = new UiAutomator2Options();
        AndroidDriver driver = new AndroidDriver(new URL(endpoint), options);

        try {
            // ... your test ...
        } finally {
            driver.quit();
        }
    }
}

```

JavaScript

Remarque : cet exemple utilise le AWS SDK pour JavaScript v3 et WebDriverIO v8+ à l'aide de Node 18+.

Une fois votre session ouverte, l'URL du point de terminaison Appium sera disponible via un champ nommé `remoteDriverEndpoint` dans la réponse à un appel à l'[GetRemoteAccessSessionAPI](#) :

```
// To get the URL
import { DeviceFarmClient, GetRemoteAccessSessionCommand } from "@aws-sdk/client-device-farm";

export async function getAppiumEndpoint() {
  const sessionArn = "arn:aws:devicefarm:us-west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/abcdef123456-1234-5678-abcd-abcdef123456/000000";

  const client = new DeviceFarmClient({ region: "us-west-2" });
  const resp = await client.send(new GetRemoteAccessSessionCommand({ arn: sessionArn }));

  const endpoint = resp?.remoteAccessSession?.endpoints?.remoteDriverEndpoint;
  if (!endpoint) throw new Error("remoteDriverEndpoint missing from response");
  return endpoint;
}

// To use the URL with WebdriverIO
import { remote } from "webdriverio";

(async () => {
  const endpoint = await getAppiumEndpoint();
  const u = new URL(endpoint);

  const driver = await remote({
    protocol: u.protocol.replace(":", ""),
    hostname: u.hostname,
    port: u.port ? Number(u.port) : (u.protocol === "https:" ? 443 : 80),
    path: u.pathname + u.search,
    capabilities: {
      platformName: "Android",
      "appium:automationName": "UiAutomator2",
      // ...other caps...
    }
  });
})();
```



```

    },
  });

  try {
    // ... your test ...
  } finally {
    await driver.deleteSession();
  }
})();

```

C#

Une fois votre session ouverte, l'URL du point de terminaison Appium sera disponible via un champ nommé `remoteDriverEndpoint` dans la réponse à un appel à l'[GetRemoteAccessSessionAPI](#) :

```

// To get the URL
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

public static class AppiumEndpointBuilder
{
    public static async Task<string> GetAppiumEndpointAsync()
    {
        var sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/000000";

        var config = new AmazonDeviceFarmConfig
        {
            RegionEndpoint = RegionEndpoint.USSouth2
        };
        using var client = new AmazonDeviceFarmClient(config);

        var resp = await client.GetRemoteAccessSessionAsync(new
        GetRemoteAccessSessionRequest { Arn = sessionArn });
        var endpoint = resp?.RemoteAccessSession?.Endpoints?.RemoteDriverEndpoint;

        if (string.IsNullOrEmpty(endpoint))
    }
}

```

```
        throw new InvalidOperationException("RemoteDriverEndpoint missing from
response");

        return endpoint;
    }
}

// To use the URL
using OpenQA.Selenium.Appium;
using OpenQA.Selenium.Appium.Android;

class Example
{
    static async Task Main()
    {
        var endpoint = await AppiumEndpointBuilder.GetAppiumEndpointAsync();

        var options = new AppiumOptions();
        options.PlatformName = "Android";
        options.AutomationName = "UiAutomator2";

        using var driver = new AndroidDriver(new Uri(endpoint), options);
        try
        {
            // ... your test ...
        }
        finally
        {
            driver.Quit();
        }
    }
}
```

Ruby

Une fois votre session ouverte, l'URL du point de terminaison Appium sera disponible via un champ nommé `remoteDriverEndpoint` dans la réponse à un appel à [l'GetRemoteAccessSessionAPI](#) :

```
# To get the URL
require 'aws-sdk-devicefarm'

def get_appium_endpoint
```

```
session_arn = "arn:aws:devicefarm:us-  
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/  
abcdef123456-1234-5678-abcd-abcdef123456/000000"  
  
client = Aws::DeviceFarm::Client.new(region: 'us-west-2')  
resp = client.get_remote_access_session(arn: session_arn)  
endpoint = resp.remote_access_session.endpoints.remote_driver_endpoint  
raise "remote_driver_endpoint missing from response" if endpoint.nil? ||  
endpoint.empty?  
  endpoint  
end  
  
# To use the URL  
require 'appium_lib_core'  
  
endpoint = get_appium_endpoint  
opts = {  
  server_url: endpoint,  
  capabilities: {  
    'platformName' => 'Android',  
    'appium:automationName' => 'UiAutomator2'  
  }  
}  
  
driver = Appium::Core.for(opts).start_driver  
begin  
  # ... your test ...  
ensure  
  driver.quit  
end
```

Consulter les journaux de votre serveur Appium

Une fois que vous avez [démarré une session Appium](#), vous pouvez consulter les journaux du serveur Appium en direct sur la console Device Farm ou les télécharger une fois la session d'accès à distance terminée. Voici les instructions pour ce faire :

Console

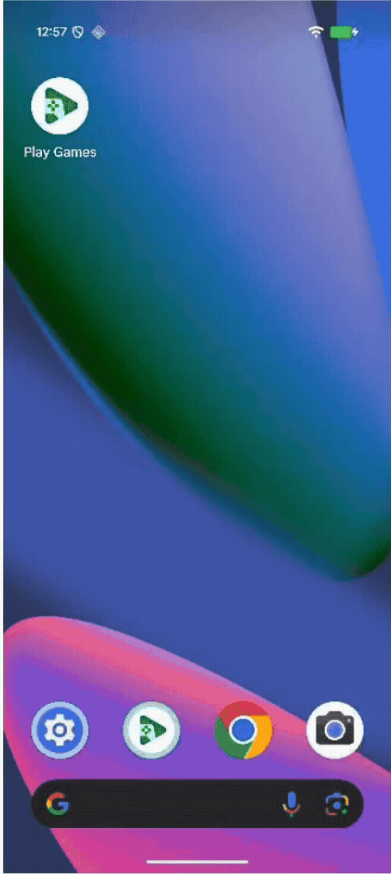
1. Dans la console Device Farm, ouvrez la session d'accès à distance pour votre appareil.

2. Démarrez une session de point de terminaison Appium avec l'appareil depuis votre IDE local ou Appium Inspector
3. Ensuite, le journal du serveur Appium apparaîtra à côté de l'appareil sur la page de session d'accès à distance, avec les « informations de session » disponibles au bas de la page sous l'appareil :

Device Farm > Mobile Device: Projects > Project: Appium endpoint demo > Session: Google Pixel 10

Google Pixel 10

Hide session information Setup Appium session Stop Session



Session information

Upload app
Upload an Android app as a .apk. No instrumentation or provisioning required.

Choose File or drop file here

Install an existing file
Install a previously uploaded application

Select a recent upload

Session ARN
arn:aws:devicefarm:us-west-2:265366432518:session:89d74780-1...

Appium endpoint URL
https://aatpg-interactive-global.us-west-2.api.aws/remote-en...

Time left
02:23:04

Device name
Google Pixel 10

OS
16

Notice
Click CTRL+M to shift focus from the mobile device screen to the Stop Session button.

Notice
To download an app from the Play Store, add your Google Account to the device. Once you do that, you will be able to see all apps in the Play Store. Note that AWS Device Farm captures video and logs of activity taking place during Remote Access session. It is recommended that you avoid entering your personal accounts on the device (for example, a personal Google account) and instead use test accounts where possible.

Back Home Recent Apps Screenshot Landscape

AWS CLI

Remarque : cet exemple utilise l'[outil de ligne de commande curl](#) pour extraire le journal de Device Farm.

Pendant ou après la session, vous pouvez utiliser l'[ListArtifacts](#) API de Device Farm pour télécharger le journal du serveur Appium.

```
$ aws devicefarm list-artifacts \
```

```
--type FILE \
--arn arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678
```

Cela affichera des résultats tels que les suivants au cours de la session :

```
{
  "artifacts": [
    {
      "arn": "arn:aws:devicefarm:us-
west-2:111122223333:artifact:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678",
      "name": "AppiumServerLogOutput",
      "type": "APPIUM_SERVER_LOG_OUTPUT",
      "extension": "",
      "url": "https://prod-us-west-2-results.s3.dualstack.us-
west-2.amazonaws.com/111122223333/12345678..."
    }
  ]
}
```

Et ce qui suit une fois la session terminée :

```
{
  "artifacts": [
    {
      "arn": "arn:aws:devicefarm:us-
west-2:111122223333:artifact:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678",
      "name": "Appium Server Output",
      "type": "APPIUM_SERVER_OUTPUT",
      "extension": "log",
      "url": "https://prod-us-west-2-results.s3.dualstack.us-
west-2.amazonaws.com/111122223333/12345678..."
    }
  ]
}
```

```
$ curl "https://prod-us-west-2-results.s3.dualstack.us-
west-2.amazonaws.com/111122223333/12345678..."
```

Cela affichera des résultats tels que les suivants :

```
info Appium Welcome to Appium v2.5.4
```

```

info Appium Non-default server args:
info Appium { address: '127.0.0.1',
info Appium   allowInsecure:
info Appium     [ 'execute_driver_script',
info Appium       'session_discovery',
info Appium       'perf_record',
info Appium       'adb_shell',
info Appium       'chromedriver_autodownload',
info Appium       'get_server_logs' ],
info Appium   keepAliveTimeout: 0,
info Appium   logNoColors: true,
info Appium   logTimestamp: true,
info Appium   longStackTrace: true,
info Appium   sessionOverride: true,
info Appium   strictCaps: true,
info Appium   useDrivers: [ 'uiautomator' ] }

```

Python

Remarque : cet exemple utilise le *requests* package tiers pour télécharger le journal, ainsi que le AWS SDK pour Python *boto3*.

Pendant ou après la session, vous pouvez utiliser l'[ListArtifacts](#) API de Device Farm pour récupérer l'URL du journal du serveur Appium, puis la télécharger.

```

import pathlib
import requests
import boto3

def download_appium_log():
    session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678
    client = boto3.client("devicefarm", region_name="us-west-2")

    # 1) List artifacts for the session (FILE artifacts), handling pagination
    artifacts = []
    token = None
    while True:
        kwargs = {"arn": session_arn, "type": "FILE"}
        if token:
            kwargs["nextToken"] = token
        resp = client.list_artifacts(**kwargs)
        artifacts.extend(resp.get("artifacts", []))

```

```

        token = resp.get("nextToken")
        if not token:
            break

    if not artifacts:
        raise RuntimeError("No artifacts found in this session")

    # Filter strictly to Appium server logs
    allowed = {"APPIUM_SERVER_OUTPUT", "APPIUM_SERVER_LOG_OUTPUT"}
    filtered = [a for a in artifacts if a.get("type") in allowed]
    if not filtered:
        raise RuntimeError("No Appium server log artifacts found (expected
APPIUM_SERVER_OUTPUT or APPIUM_SERVER_LOG_OUTPUT)")

    # Prefer the final 'OUTPUT' log, else the live 'LOG_OUTPUT'
    chosen = (next((a for a in filtered if a.get("type") == "APPIUM_SERVER_OUTPUT"),
None)
            or next((a for a in filtered if a.get("type") ==
"APPIUM_SERVER_LOG_OUTPUT"), None))

    url = chosen["url"]
    ext = chosen.get("extension") or "log"
    out = pathlib.Path(f"./appium_server_log.{ext}")

    # 2) Download the artifact
    with requests.get(url, stream=True) as r:
        r.raise_for_status()
        with open(out, "wb") as fh:
            for chunk in r.iter_content(chunk_size=1024 * 1024):
                if chunk:
                    fh.write(chunk)

    print(f"Saved Appium server log to: {out.resolve()}")

download_appium_log()

```

Cela affichera des résultats tels que les suivants :

```

info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', allowInsecure: [ 'execute_driver_script', ... ],
  useDrivers: [ 'uiautomator' ] }

```

Java

Remarque : cet exemple utilise le AWS SDK pour Java v2 *HttpClient* et pour télécharger le journal. Il est compatible avec les versions 11 et supérieures du JDK.

Pendant ou après la session, vous pouvez utiliser l'[ListArtifacts](#) API de Device Farm pour récupérer l'URL du journal du serveur Appium, puis la télécharger.

```
import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.nio.file.Path;
import java.time.Duration;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import software.amazon.awssdk.services.devicefarm.model.Artifact;
import software.amazon.awssdk.services.devicefarm.model.ArtifactCategory;
import software.amazon.awssdk.services.devicefarm.model.ListArtifactsRequest;
import software.amazon.awssdk.services.devicefarm.model.ListArtifactsResponse;

public class AppiumLogDownloader {

    public static void main(String[] args) throws Exception {
        String sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789abcdef";

        try (DeviceFarmClient client = DeviceFarmClient.builder()
            .region(Region.US_WEST_2)
            .build()) {

            // 1) List artifacts for the session (FILE artifacts) with pagination
            List<Artifact> all = new ArrayList<>();
            String token = null;
            do {
                ListArtifactsRequest.Builder b = ListArtifactsRequest.builder()
                    .arn(sessionArn)
                    .type(ArtifactCategory.FILE);
```



```
        if (token != null) b.nextToken(token);
        ListArtifactsResponse page = client.listArtifacts(b.build());
        all.addAll(page.artifacts());
        token = page.nextToken();
    } while (token != null && !token.isBlank());

    // Filter strictly to Appium logs
    List<Artifact> filtered = all.stream()
        .filter(a -> {
            String t = a.typeAsString();
            return "APPIUM_SERVER_OUTPUT".equals(t) ||
"APPIUM_SERVER_LOG_OUTPUT".equals(t);
        })
        .toList();

    if (filtered.isEmpty()) {
        throw new RuntimeException("No Appium server log artifacts found
(expected APPIUM_SERVER_OUTPUT or APPIUM_SERVER_LOG_OUTPUT).");
    }

    // Prefer OUTPUT; else LOG_OUTPUT
    Artifact chosen = filtered.stream()
        .filter(a -> "APPIUM_SERVER_OUTPUT".equals(a.typeAsString()))
        .findFirst()
        .orElseGet(() -> filtered.stream()
            .filter(a ->
"APPIUM_SERVER_LOG_OUTPUT".equals(a.typeAsString()))
            .findFirst()
            .get());

    String url = chosen.url();
    String ext = (chosen.extension() == null ||
chosen.extension().isBlank()) ? "log" : chosen.extension();
    Path out = Path.of("appium_server_log." + ext);

    // 2) Download the artifact with HttpClient
    HttpClient http = HttpClient.newBuilder()
        .connectTimeout(Duration.ofSeconds(10))
        .build();

    HttpRequest get = HttpRequest.newBuilder(URI.create(url))
        .timeout(Duration.ofMinutes(5))
        .GET()
        .build();
```

```

        HttpResponse<Path> resp = http.send(get,
        HttpResponse.BodyHandlers.ofFile(out));
        if (resp.statusCode() / 100 != 2) {
            throw new IOException("Failed to download log, HTTP " +
        resp.statusCode());
        }
        System.out.println("Saved Appium server log to: " +
        out.toAbsolutePath());
    }
}
}

```

Cela affichera des résultats tels que les suivants :

```

info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', ..., useDrivers: [ 'uiautomator' ] }

```

JavaScript

Remarque : cet exemple utilise le AWS SDK pour JavaScript (v3) et Node 18+ *fetch* pour télécharger le journal.

Pendant ou après la session, vous pouvez utiliser l'[ListArtifacts](#) API de Device Farm pour récupérer l'URL du journal du serveur Appium, puis la télécharger.

```

import { DeviceFarmClient, ListArtifactsCommand } from "@aws-sdk/client-device-farm";
import { createWriteStream } from "fs";
import { pipeline } from "stream";
import { promisify } from "util";

const pipe = promisify(pipeline);
const client = new DeviceFarmClient({ region: "us-west-2" });

const sessionArn = "arn:aws:devicefarm:us-west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789abcdef";

// 1) List artifacts for the session (FILE artifacts), handling pagination
const artifacts = [];
let nextToken;

```

```

do {
  const page = await client.send(new ListArtifactsCommand({
    arn: sessionArn,
    type: "FILE",
    nextToken
  }));
  artifacts.push...(page.artifacts ?? []);
  nextToken = page.nextToken;
} while (nextToken);

if (!artifacts.length) throw new Error("No artifacts found");

// Strict filter to Appium logs
const filtered = (artifacts ?? []).filter(a =>
  a.type === "APPIUM_SERVER_OUTPUT" || a.type === "APPIUM_SERVER_LOG_OUTPUT"
);
if (!filtered.length) {
  throw new Error("No Appium server log artifacts found (expected
    APPIUM_SERVER_OUTPUT or APPIUM_SERVER_LOG_OUTPUT).");
}

// Prefer OUTPUT; else LOG_OUTPUT
const chosen =
  filtered.find(a => a.type === "APPIUM_SERVER_OUTPUT") ??
  filtered.find(a => a.type === "APPIUM_SERVER_LOG_OUTPUT");

const url = chosen.url;
const ext = chosen.extension || "log";
const outPath = `./appium_server_log.${ext}`;

// 2) Download the artifact
const resp = await fetch(url);
if (!resp.ok) {
  throw new Error(`Failed to download log: ${resp.status} ${await
    resp.text().catch(()=>"")}`);
}
await pipe(resp.body, createWriteStream(outPath));
console.log("Saved Appium server log to:", outPath);

```

Cela affichera des résultats tels que les suivants :

```

info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:

```

```
info Appium { address: '127.0.0.1', allowInsecure: [ 'execute_driver_script', ... ],
  useDrivers: [ 'uiautomator' ] }
```

C#

Remarque : cet exemple utilise le AWS SDK pour .NET *HttpClient* et permet de télécharger le journal.

Pendant ou après la session, vous pouvez utiliser l'[ListArtifacts](#) API de Device Farm pour récupérer l'URL du journal du serveur Appium, puis la télécharger.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Net.Http;
using System.Threading.Tasks;
using System.Linq;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

class AppiumLogDownloader
{
    static async Task Main()
    {
        var sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678

        using var client = new AmazonDeviceFarmClient(RegionEndpoint.USSouth2);

        // 1) List artifacts for the session (FILE artifacts), handling pagination
        var all = new List<Artifact>();
        string? token = null;
        do
        {
            var page = await client.ListArtifactsAsync(new ListArtifactsRequest
            {
                Arn = sessionArn,
                Type = ArtifactCategory.FILE,
                NextToken = token
            });
            if (page.Artifacts != null) all.AddRange(page.Artifacts);
            token = page.NextToken;
        }
```

```

    } while (!string.IsNullOrEmpty(token));

    if (all.Count == 0)
        throw new Exception("No artifacts found");

    // Strict filter to Appium logs
    var filtered = all.Where(a =>
        a.Type == "APPIUM_SERVER_OUTPUT" || a.Type ==
        "APPIUM_SERVER_LOG_OUTPUT").ToList();

    if (filtered.Count == 0)
        throw new Exception("No Appium server log artifacts found (expected
        APPIUM_SERVER_OUTPUT or APPIUM_SERVER_LOG_OUTPUT).");

    // Prefer OUTPUT; else LOG_OUTPUT
    var chosen = filtered.FirstOrDefault(a => a.Type == "APPIUM_SERVER_OUTPUT")
        ?? filtered.First(a => a.Type == "APPIUM_SERVER_LOG_OUTPUT");

    var url = chosen.Url;
    var ext = string.IsNullOrEmpty(chosen.Extension) ? "log" :
    chosen.Extension;
    var outPath = $"./appium_server_log.{ext}";

    // 2) Download the artifact
    using var http = new HttpClient();
    using var resp = await http.GetAsync(url,
    HttpCompletionOption.ResponseHeadersRead);
    resp.EnsureSuccessStatusCode();
    await using (var fs = File.Create(outPath))
    {
        await resp.Content.CopyToAsync(fs);
    }
    Console.WriteLine($"Saved Appium server log to:
    {Path.GetFullPath(outPath)}");
}
}

```

Cela affichera des résultats tels que les suivants :

```

info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', ..., useDrivers: [ 'uiautomator' ] }

```

Ruby

Remarque : cet exemple utilise le AWS SDK pour Ruby *Net::HTTP* et permet de télécharger le journal.

Pendant ou après la session, vous pouvez utiliser l'[ListArtifacts](#) API de Device Farm pour récupérer l'URL du journal du serveur Appium, puis la télécharger.

```
require "aws-sdk-devicefarm"
require "net/http"
require "uri"

client = Aws::DeviceFarm::Client.new(region: "us-west-2")
session_arn = "arn:aws:devicefarm:us-west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789abcdef"

# 1) List artifacts for the session (FILE artifacts), handling pagination
artifacts = []
token = nil
loop do
  page = client.list_artifacts(arn: session_arn, type: "FILE", next_token: token)
  artifacts.concat(page.artifacts || [])
  token = page.next_token
  break if token.nil? || token.empty?
end

raise "No artifacts found" if artifacts.empty?

# Strict filter to Appium logs
filtered = (artifacts || []).select { |a| ["APPIUM_SERVER_OUTPUT",
  "APPIUM_SERVER_LOG_OUTPUT"].include?(a.type) }
raise "No Appium server log artifacts found (expected APPIUM_SERVER_OUTPUT or
  APPIUM_SERVER_LOG_OUTPUT)." if filtered.empty?

# Prefer OUTPUT; else LOG_OUTPUT
chosen = filtered.find { |a| a.type == "APPIUM_SERVER_OUTPUT" } ||
  filtered.find { |a| a.type == "APPIUM_SERVER_LOG_OUTPUT" }

url = chosen.url
ext = (chosen.extension && !chosen.extension.empty?) ? chosen.extension : "log"
out_path = "./appium_server_log.#{ext}"

# 2) Download the artifact
```

```
uri = URI.parse(url)
Net::HTTP.start(uri.host, uri.port, use_ssl: (uri.scheme == "https")) do |http|
  req = Net::HTTP::Get.new(uri)
  http.request(req) do |resp|
    raise "Failed GET: #{resp.code} #{resp.body}" unless resp.code.to_i / 100 == 2
    File.open(out_path, "wb") { |f| resp.read_body { |chunk| f.write(chunk) } }
  end
end
puts "Saved Appium server log to: #{File.expand_path(out_path)}"
```

Cela affichera des résultats tels que les suivants :

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', allowInsecure: [ 'execute_driver_script', ... ],
  useDrivers: [ 'uiautomator' ] }
```

Fonctionnalités et commandes Appium prises en charge

Le point de terminaison Appium de Device Farm prend en charge la plupart des commandes et fonctionnalités souhaitées que vous utilisez sur les appareils locaux, à quelques exceptions près. Les listes suivantes indiquent les fonctionnalités et les commandes actuellement non prises en charge. Si vos tests ne peuvent pas s'exécuter comme prévu en raison d'une fonctionnalité limitée, veuillez ouvrir un dossier d'assistance pour obtenir des conseils supplémentaires.

Fonctionnalités prises en charge

Lorsque vous créez une session Appium sur Device Farm, nous vous recommandons de disposer d'un ensemble distinct de fonctionnalités qui excluent les fonctionnalités spécifiques à votre appareil local. Sur Device Farm, la création de session peut échouer si certaines fonctionnalités non prises en charge sont définies. Cela inclut des fonctionnalités spécifiques à l'appareil telles que `udid` et `platformVersion`. En outre, certaines fonctionnalités liées à `ChromeDriver` Android et `WebDriverAgent` iOS ne sont pas prises en charge, de même que celles qui ne sont prises en charge que sur les émulateurs et les simulateurs.

Commandes prises en charge

La plupart des commandes Appium qui s'exécutent correctement sur de vrais appareils Android et iOS s'exécuteront comme prévu sur Device Farm, avec les exclusions suivantes :

Commandes de l'appareil Appium () **/appium/device**

- `install_app`
- `finger_print`
- `send_sms`
- `gsm_call`
- `gsm_signal`
- `gsm_voice`
- `power_ac`
- `power_capacity`
- `network_speed`
- `shake`

Méthodes et scripts d'exécution Appium () **/execute**

- `installApp`
- `execEmuConsoleCommand`
- `fingerprint`
- `gsmCall`
- `gsmSignal`
- `sendSms`
- `gsmVoice`
- `powerAC`
- `powerCapacity`
- `networkSpeed`
- `sensorSet`
- `injectEmulatorCameraImage`
- `isGpsEnabled`
- `shake`
- `clearApp`
- `clearKeychains`

- `configureLocalization`
- `enrollBiometric`
- `getPasteboard`
- `installXCTestBundle`
- `listXCTestBundles`
- `listXCTestsInTestBundle`
- `runXCTest`
- `sendBiometricMatch`
- `setPasteboard`
- `setPermission`
- `startAudioRecording`
- `startLogsBroadcast`
- `startRecordingScreen`
- `startScreenStreaming`
- `startXCTestScreenRecording`
- `stopAudioRecording`
- `stopLogsBroadcast`
- `stopRecordingScreen`
- `stopScreenStreaming`
- `stopXCTestScreenRecording`
- `updateSafariPreferences`

Appareils privés dans AWS Device Farm

Un appareil privé est un appareil mobile physique qu'AWS Device Farm déploie en votre nom dans un centre de données Amazon. Cet appareil est exclusif à votre AWS compte.

Note

Actuellement, les appareils privés ne sont disponibles que dans la région Ouest des AWS États-Unis (Oregon) (us-west-2).

Si vous disposez d'un parc d'appareils privés, vous pouvez créer des sessions d'accès à distance et planifier des exécutions de test avec vos appareils privés. Pour de plus amples informations, veuillez consulter [Création d'un test ou démarrage d'une session d'accès à distance dans AWS Device Farm](#). Vous pouvez également créer des profils d'instance pour contrôler le comportement de vos appareils privés lors d'une session d'accès à distance ou d'une série de tests. Pour de plus amples informations, veuillez consulter [Création d'un profil d'instance dans AWS Device Farm](#). Vous pouvez éventuellement demander que certains appareils privés Android soient déployés en tant qu'appareils rootés.

Vous pouvez également créer un service de point de terminaison Amazon Virtual Private Cloud pour tester les applications privées auxquelles votre entreprise a accès, mais qui ne sont pas accessibles via Internet. Par exemple, vous pouvez avoir une application web s'exécutant dans votre VPC que vous souhaitez tester sur les appareils mobiles. Pour de plus amples informations, veuillez consulter [Utilisation des services de point de terminaison Amazon VPC avec Device Farm - Legacy \(non recommandé\)](#).

Si vous souhaitez utiliser un parc d'appareils privés, [contactez-nous](#). L'équipe Device Farm doit travailler avec vous pour configurer et déployer une flotte d'appareils privés pour votre AWS compte.

Rubriques

- [Création d'un profil d'instance dans AWS Device Farm](#)
- [Demandez des appareils privés supplémentaires dans AWS Device Farm](#)
- [Création d'un test ou démarrage d'une session d'accès à distance dans AWS Device Farm](#)
- [Sélection d'appareils privés dans un pool d'appareils dans AWS Device Farm](#)
- [Ignorer la nouvelle signature d'une application sur des appareils privés dans AWS Device Farm](#)

- [Amazon VPC dans toutes les AWS régions d'AWS Device Farm](#)
- [Résiliation d'appareils privés dans Device Farm](#)

Création d'un profil d'instance dans AWS Device Farm

Vous pouvez configurer un parc qui contient un ou plusieurs appareils privés. Ces appareils sont dédiés à votre compte AWS . Une fois que vous avez configuré les appareils, vous pouvez, le cas échéant, créer un ou plusieurs profils d'instances pour eux. Les profils d'instances peuvent vous aider à automatiser les séries de tests et toujours appliquer les mêmes paramètres aux instances d'appareils. Les profils d'instances peuvent également vous aider à contrôler le comportement de la session d'accès à distance. Pour plus d'informations sur les appareils privés dans Device Farm, consultez [Appareils privés dans AWS Device Farm](#).

Pour créer une instance

1. Ouvrez la console Device Farm à l'adresse <https://console.aws.amazon.com/devicefarm/>.
2. Sur le panneau de navigation de Device Farm, choisissez Mobile Device Testing, puis Private devices.
3. Choisissez Instance profiles (Profils d'instance).
4. Choisissez Créer un profil d'instance.
5. Saisissez le nom du profil d'instance.

Create a new instance profile ✕

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

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

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

☐ Reboot after use

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

☐ Package cleanup after use

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

[+ Add new](#)

Cancel Save

6. (Facultatif) Entrez une description pour le profil d'instance.
7. (Facultatif) Modifiez l'un des paramètres suivants pour spécifier les actions que Device Farm doit effectuer sur un appareil à la fin de chaque test ou de chaque session :
 - Redémarrer après utilisation : pour redémarrer l'appareil, cochez cette case. Par défaut, cette case est décochée (`false`).
 - Nettoyage des packages : pour supprimer tous les packages d'applications que vous avez installés sur l'appareil, cochez cette case. Par défaut, cette case est décochée (`false`). Pour conserver toutes les packages d'applications que vous avez installés sur l'appareil, laissez cette case décochée.

- Exclure les packages du nettoyage : pour ne conserver que les packages d'applications sélectionnés sur l'appareil, cochez la case Nettoyage du package, puis cliquez sur Ajouter un nouveau. Pour le nom du package, saisissez le nom complet du package d'applications que vous souhaitez conserver sur l'appareil (par exemple, `com.test.example`). Pour conserver plus de packages d'applications sur l'appareil, choisissez Add new (Ajouter nouveau), puis saisissez le nom complet de chaque package.

8. Choisissez Enregistrer.

Demandez des appareils privés supplémentaires dans AWS Device Farm

Dans AWS Device Farm, vous pouvez demander que des instances d'appareils privés supplémentaires soient ajoutées à votre flotte. Vous pouvez également consulter et modifier les paramètres des instances d'appareils privés existantes dans votre parc. Pour plus d'informations sur les appareils privés, consultez [Appareils privés dans AWS Device Farm](#).

Pour demander des appareils privés supplémentaires ou modifier leurs paramètres

1. Ouvrez la console Device Farm à l'adresse <https://console.aws.amazon.com/devicefarm/>.
2. Sur le panneau de navigation de Device Farm, choisissez Mobile Device Testing, puis Private devices.
3. Choisissez Device instances (Instances d'appareils). L'onglet Device Instances (Instances d'appareils) affiche une table des appareils privés qui se trouvent dans votre parc. Pour rechercher ou filtrer rapidement le tableau, entrez les termes de recherche dans la barre de recherche située au-dessus des colonnes.
4. Pour demander une nouvelle instance d'appareil privée, choisissez Demander une instance d'appareil ou [contactez-nous](#). Les appareils privés nécessitent une configuration supplémentaire avec l'aide de l'équipe Device Farm.
5. Dans le tableau des instances de terminal, choisissez l'option de bascule située à côté de l'instance que vous souhaitez consulter ou gérer, puis choisissez Modifier.

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

6. Pour associer un profil d'instance à l'instance de terminal, sélectionnez-le dans la liste déroulante Profil. Il peut être utile de joindre un profil d'instance si vous souhaitez toujours exclure un package d'application spécifique des tâches de nettoyage, par exemple. Pour plus d'informations sur l'utilisation de profils d'instance avec des appareils, consultez [Création d'un profil d'instance dans AWS Device Farm](#).
7. (Facultatif) Sous Étiquettes, choisissez Add new (Ajouter nouveau) pour ajouter une étiquette à l'instance d'appareil. Les étiquettes peuvent vous aider à classer vos appareils et trouver des appareils spécifiques plus facilement.
8. Choisissez Enregistrer.

Création d'un test ou démarrage d'une session d'accès à distance dans AWS Device Farm

Dans AWS Device Farm, après avoir configuré un parc d'appareils privé, vous pouvez créer des tests ou démarrer des sessions d'accès à distance avec un ou plusieurs appareils privés de votre parc.

Pour plus d'informations sur les appareils privés, consultez [Appareils privés dans AWS Device Farm](#).

Pour créer un test ou démarrer une session d'accès à distance

1. Ouvrez la console Device Farm à l'adresse <https://console.aws.amazon.com/devicefarm/>.
2. Sur le panneau de navigation de Device Farm, choisissez Mobile Device Testing, puis Projects.
3. Choisissez un projet existant dans la liste ou créez-en un nouveau. Pour créer un nouveau projet, choisissez Nouveau projet, entrez un nom pour le projet, puis choisissez Soumettre.
4. Effectuez l'une des actions suivantes :
 - Pour créer une série de tests, choisissez Automated tests (Tests automatisés), puis choisissez Create a new run (Créer une exécution). L'assistant vous décrit la marche à suivre pour créer l'exécution. Pour l'étape Select devices, vous pouvez modifier un pool d'appareils existant ou en créer un nouveau qui inclut uniquement les appareils privés que l'équipe Device Farm a configurés et associés à votre AWS compte. Pour de plus amples informations, veuillez consulter [the section called "Création d'un pool d'appareils privé"](#).
 - Pour démarrer une session d'accès à distance, choisissez Remote access (Accès à distance), puis choisissez Start a new session (Démarrer une nouvelle session). Sur la page Choose a device, sélectionnez Private device instances only pour limiter la liste aux seuls appareils privés que l'équipe Device Farm a configurés et associés à votre AWS compte. Ensuite, choisissez l'appareil auquel vous souhaitez accéder, saisissez un nom pour la session d'accès à distance, puis choisissez Confirm and start session (Confirmer et démarrer la session).

Create a new remote session

Choose a device

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

☒ Private device instances only

☐ Show available devices only

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

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

< 1 2 >

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

Sélection d'appareils privés dans un pool d'appareils dans AWS Device Farm

Pour utiliser des appareils privés lors de votre test, vous pouvez créer un pool d'appareils qui sélectionne vos appareils privés. Les pools d'appareils vous permettent de sélectionner des appareils privés principalement par le biais de trois types de règles de pool d'appareils :

1. Règles basées sur l'ARN de l'appareil
2. Règles basées sur l'étiquette de l'instance de l'appareil
3. Règles basées sur l'ARN de l'instance de l'appareil

Dans les sections suivantes, chaque type de règle et ses cas d'utilisation sont décrits en détail. Vous pouvez utiliser la console Device Farm, l'interface de ligne de AWS commande (AWS CLI) ou l'API Device Farm pour créer ou modifier un pool d'appareils avec des appareils privés à l'aide de ces règles.

Rubriques

- [ARN de l'appareil](#)
- [Étiquettes des instances de l'appareil](#)
- [ARN de l'instance](#)
- [Création d'un pool d'appareils privés avec des appareils privés \(console\)](#)
- [Création d'un pool d'appareils privés avec des appareils privés \(AWS CLI\)](#)

- [Création d'un pool d'appareils privés avec des appareils privés \(API\)](#)

ARN de l'appareil

Un ARN de périphérique est un identifiant représentant un type de périphérique plutôt qu'une instance de périphérique physique spécifique. Un type d'appareil est défini par les attributs suivants :

- L'identifiant de flotte de l'appareil
- Le fabricant OEM de l'appareil
- Le numéro de modèle de l'appareil
- Version du système d'exploitation de l'appareil
- L'état de l'appareil qui indique s'il est rooté ou non

De nombreuses instances de périphérique physique peuvent être représentées par un seul type de périphérique, chaque instance de ce type ayant les mêmes valeurs pour ces attributs. Par exemple, si vous disposiez de trois *Apple iPhone 13* appareils utilisant la version iOS *16.1.0* dans votre parc privé, chaque appareil partagerait le même ARN. Si des appareils étaient ajoutés ou retirés de votre parc avec ces mêmes attributs, l'ARN de l'appareil continuerait de représenter les appareils disponibles dans votre parc pour ce type d'appareil.

L'ARN des appareils est le moyen le plus fiable de sélectionner des appareils privés pour un pool d'appareils, car il permet au pool d'appareils de continuer à sélectionner des appareils quelles que soient les instances spécifiques que vous avez déployées à un moment donné. Les instances individuelles d'appareils privés peuvent subir des pannes matérielles, ce qui incite Device Farm à les remplacer automatiquement par de nouvelles instances opérationnelles du même type d'appareil. Dans ces scénarios, la règle ARN des appareils garantit que votre pool d'appareils peut continuer à sélectionner des appareils en cas de panne matérielle.

Lorsque vous utilisez une règle d'ARN pour les appareils privés de votre pool d'appareils et que vous planifiez un test avec ce pool, Device Farm vérifie automatiquement quelles instances d'appareils privés sont représentées par cet ARN d'appareil. Parmi les instances actuellement disponibles, l'une d'entre elles sera affectée à l'exécution de votre test. Si aucune instance n'est actuellement disponible, Device Farm attendra que la première instance disponible de cet ARN de périphérique soit disponible et l'attribuera pour exécuter votre test.

Étiquettes des instances de l'appareil

Une étiquette d'instance de périphérique est un identifiant textuel que vous pouvez associer en tant que métadonnées à une instance de périphérique. Vous pouvez associer plusieurs étiquettes à chaque instance de terminal et la même étiquette à plusieurs instances de terminal. Pour plus d'informations sur l'ajout, la modification ou la suppression d'étiquettes d'appareils sur les instances d'appareils, consultez [la section Gestion des appareils privés](#).

L'étiquette d'instance d'appareil peut être un moyen efficace de sélectionner des appareils privés pour un pool d'appareils car, si plusieurs instances d'appareils portent la même étiquette, elle permet au pool d'appareils de sélectionner l'un d'entre eux pour votre test. Si l'ARN de l'appareil n'est pas une bonne règle pour votre cas d'utilisation (par exemple, si vous souhaitez sélectionner des appareils de plusieurs types d'appareils, ou si vous souhaitez sélectionner un sous-ensemble de tous les appareils d'un type d'appareil), les étiquettes d'instance de périphérique peuvent vous permettre de sélectionner plusieurs appareils pour votre pool d'appareils avec une plus grande granularité. Les instances individuelles d'appareils privés peuvent subir des pannes matérielles, ce qui incite Device Farm à les remplacer automatiquement par de nouvelles instances opérationnelles du même type d'appareil. Dans ces scénarios, l'instance de périphérique de remplacement ne conservera aucune métadonnée d'étiquette d'instance du périphérique remplacé. Ainsi, si vous appliquez la même étiquette d'instance de périphérique à plusieurs instances de périphérique, la règle d'étiquette d'instance de périphérique garantit que votre pool d'appareils peut continuer à sélectionner des instances de périphériques en cas de panne matérielle.

Lorsque vous utilisez une règle d'étiquette d'instance d'appareil pour les appareils privés de votre pool d'appareils et que vous planifiez un test avec ce pool, Device Farm vérifie automatiquement quelles instances privées sont représentées par cette étiquette d'instance d'appareil et, parmi ces instances, sélectionne au hasard une instance disponible pour exécuter votre test. Si aucune n'est disponible, Device Farm sélectionnera au hasard une instance d'appareil portant le nom d'instance de périphérique pour exécuter votre test et mettra le test en file d'attente pour qu'il soit exécuté sur l'appareil une fois qu'il sera disponible.

ARN de l'instance

L'ARN d'une instance de périphérique est un identifiant représentant une instance physique de périphérique bare metal déployée dans un parc privé. Par exemple, si vous avez trois *iPhone 13* appareils sous système d'exploitation *15.0.0* dans votre parc privé, alors que chaque appareil partage le même ARN, chaque appareil aura également son propre ARN d'instance représentant cette instance uniquement.

L'ARN de l'instance de périphérique est le moyen le moins robuste de sélectionner des appareils privés pour un pool d'appareils et n'est recommandé que si les étiquettes de l'appareil ARNs et de l'instance d'appareil ne correspondent pas à votre cas d'utilisation. ARNs Les instances de périphériques sont souvent utilisées comme règles pour les pools de périphériques lorsqu'une instance de périphérique spécifique est configurée de manière unique et spécifique comme condition préalable à votre test et si cette configuration doit être connue et vérifiée avant que le test ne soit effectué sur elle. Les instances individuelles d'appareils privés peuvent subir des pannes matérielles, ce qui incite Device Farm à les remplacer automatiquement par de nouvelles instances opérationnelles du même type d'appareil. Dans ces scénarios, l'instance de périphérique de remplacement aura un ARN d'instance de périphérique différent de celui du périphérique remplacé. Par conséquent, si vous utilisez une instance d'appareils ARNs pour votre pool d'appareils, vous devrez modifier manuellement la définition des règles de votre pool d'appareils pour passer de l'ancien ARN au nouvel ARN. Si vous devez préconfigurer manuellement l'appareil pour le tester, cela peut constituer un flux de travail efficace (par rapport à un appareil ARNs). Pour les tests à grande échelle, il est recommandé d'essayer d'adapter ces cas d'utilisation pour qu'ils fonctionnent avec les étiquettes d'instances de périphériques et, si possible, de préconfigurer plusieurs instances de périphériques pour les tests.

Lorsque vous utilisez une règle ARN d'instance d'appareil pour les appareils privés de votre pool d'appareils et que vous planifiez un test avec ce pool, Device Farm attribue automatiquement ce test à cette instance d'appareil. Si cette instance d'appareil n'est pas disponible, Device Farm mettra en file d'attente le test sur l'appareil dès qu'il sera disponible.

Création d'un pool d'appareils privés avec des appareils privés (console)

Lorsque vous créez une série de tests, vous pouvez créer un groupe d'appareils pour la série de tests et vous assurer que le groupe inclut uniquement vos appareils privés.

Note

Lorsque vous créez un pool d'appareils avec des appareils privés dans la console, vous ne pouvez utiliser que l'une des trois règles disponibles pour sélectionner des appareils privés. Si vous souhaitez créer un pool de périphériques contenant plusieurs types de règles pour les appareils privés (par exemple, des pools de périphériques contenant des règles pour les appareils ARNs et les instances de périphériques ARNs), vous devez créer le pool via la CLI ou l'API.

1. Ouvrez la console Device Farm à l'adresse <https://console.aws.amazon.com/devicefarm/>.
2. Sur le panneau de navigation de Device Farm, choisissez Mobile Device Testing, puis Projects.
3. Choisissez un projet existant dans la liste ou créez-en un nouveau. Pour créer un nouveau projet, choisissez Nouveau projet, entrez un nom pour le projet, puis choisissez Soumettre.
4. Choisissez Paramètres du projet, puis accédez à l'onglet Pools de périphériques.
5. Choisissez Créer un pool d'appareils, puis entrez un nom et une description facultative pour votre pool d'appareils.
 - a. Pour utiliser les règles ARN des appareils pour votre pool d'appareils, choisissez Créer un pool d'appareils statique, puis sélectionnez les types d'appareils spécifiques dans la liste que vous souhaitez utiliser dans le pool d'appareils. Ne sélectionnez pas Instances d'appareils privés uniquement parce que cette option entraîne la création du pool d'appareils avec les règles ARN des instances de périphériques (au lieu des règles ARN des appareils).

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
------	--------	----------	----	-------------	-------------	--------

Cancel Create

- b. Pour utiliser les règles d'étiquetage des instances d'appareils pour votre pool d'appareils, choisissez Create dynamic Device Pool. Ensuite, pour chaque étiquette que vous souhaitez utiliser dans le pool d'appareils, choisissez Ajouter une règle. Pour chaque règle, choisissez « Étiquettes d'instance » comme « Field, choisissez « Contient » comme « Operator, et spécifiez l'étiquette d'instance de périphérique de votre choix en tant que 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. Pour utiliser les règles ARN des instances d'appareils pour votre pool d'appareils, choisissez **Create static Device Pool**, puis sélectionnez **Private device instances only** pour limiter la liste des appareils aux seules instances d'appareils privés que Device Farm a associées à votre AWS compte.

Create device pool

Name: MyPrivateDevicePool

Description - optional: Enter a short description for your device pool

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

☐ Create dynamic device pool ☒ Create static device pool

☒ See private device instances only

Mobile devices (0/92)

Find devices by attribute

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

Cancel Create

6. Choisissez **Créer**.

Création d'un pool d'appareils privés avec des appareils privés (AWS CLI)

- Exécutez la commande [create-device-pool](#).

Pour plus d'informations sur l'utilisation de Device Farm avec le AWS CLI, consultez [AWS CLI référence](#).

Création d'un pool d'appareils privés avec des appareils privés (API)

- Appelez l'API [CreateDevicePool](#).

Pour plus d'informations sur l'utilisation de l'API Device Farm, consultez [Automatiser Device Farm](#).

Ignorer la nouvelle signature d'une application sur des appareils privés dans AWS Device Farm

La signature d'applications est un processus qui consiste à signer numériquement un package d'application (par exemple, [APK](#), [IPA](#)) à l'aide d'une clé privée avant de l'installer sur un appareil ou de le publier sur un magasin d'applications tel que le Google Play Store ou l'App Store d'Apple. Pour rationaliser les tests en réduisant le nombre de signatures et de profils nécessaires et en renforçant la sécurité des données sur les appareils distants, AWS Device Farm signera à nouveau votre application une fois qu'elle aura été téléchargée sur le service.

Une fois que vous avez chargé votre application sur AWS Device Farm, le service génère une nouvelle signature pour l'application à l'aide de ses propres certificats de signature et profils d'approvisionnement. Ce processus remplace la signature d'application d'origine par la signature d'AWS Device Farm. L'application resignée est ensuite installée sur les appareils de test fournis par AWS Device Farm. La nouvelle signature permet à l'application d'être installée et exécutée sur ces appareils sans avoir besoin des certificats du développeur d'origine.

Sur iOS, nous remplaçons le profil de provisionnement intégré par un profil générique et nous signons à nouveau l'application. Si vous les fournissez, nous ajouterons des données auxiliaires au package de l'application avant l'installation afin que les données soient présentes dans le sandbox de votre application. La nouvelle signature de l'application iOS entraîne la suppression de tous les droits.

Sur Android, nous signons à nouveau l'application. Cela peut perturber les fonctionnalités qui dépendent de la signature de l'application, telles que l'API Google Maps pour Android. Cela peut également déclencher une détection anti-piratage et anti-altération disponible sur des produits tels que DexGuard. Pour les tests intégrés, nous pouvons modifier le manifeste afin d'inclure les autorisations requises pour capturer et enregistrer des captures d'écran.

Lorsque vous utilisez des appareils privés, vous pouvez ignorer l'étape au cours de laquelle AWS Device Farm signe à nouveau votre application. Cela est différent des appareils publics, où Device Farm signe toujours à nouveau votre application sur les plateformes Android et iOS.

Vous pouvez ignorer la resignature d'applications lorsque vous créez une session d'accès à distance ou une série de tests. Cela peut être utile si les fonctionnalités de votre application sont interrompues lorsque Device Farm la signe à nouveau. Par exemple, les notifications push peuvent ne pas fonctionner après la resignature. Pour plus d'informations sur les modifications apportées par Device Farm lorsqu'il teste votre application, consultez la page [AWS Device Farm FAQs](#) ou [Apps](#).

Pour ignorer la nouvelle signature d'une application dans le cadre d'un test, sélectionnez Ignorer la nouvelle signature d'une application sous Configuration supplémentaire. Cette option n'est disponible que pour les appareils privés.

▼ **Additional configuration**

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

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

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

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

Note

Si vous utilisez le XCTest framework, l'option Ignorer la nouvelle signature de l'application n'est pas disponible. Pour de plus amples informations, veuillez consulter [Intégration de Device Farm à XCTest pour iOS](#).

Les étapes complémentaires pour la configuration des paramètres de signature de vos applications varient selon que vous utilisiez des appareils privés Android ou iOS.

Ignorer la nouvelle signature d'une application sur les appareils Android

Si vous testez votre application sur un appareil privé Android, sélectionnez Skip app re-signing (Ignorer la resignature d'application) lorsque vous créez votre exécution de test ou votre session d'accès à distance. Aucune autre configuration n'est requise.

Ignorer la nouvelle signature d'une application sur les appareils iOS

Apple exige que vous signiez une application à tester avant de charger celle-ci sur un appareil. Pour les appareils iOS, vous avez deux options pour signer votre application.

- Si vous utilisez un profil de développeur interne (entreprise), vous pouvez passer directement à la section suivante, [the section called “Créez une session d'accès à distance pour faire confiance à votre application”](#).
- Si vous utilisez un profil de développement d'applications iOS ad hoc, vous devez d'abord enregistrer l'appareil avec votre compte développeur Apple, puis mettre à jour votre profil de mise en service pour inclure l'appareil privé. Vous devez ensuite resigner votre application avec le profil de mise en service que vous avez mis à jour. Vous pouvez ensuite exécuter votre application resignée dans Device Farm.

Pour enregistrer un appareil avec un profil de développement d'applications iOS ad hoc

1. Connectez-vous à votre compte développeur Apple.
2. Accédez à la section Certificats et profils de la console. IDs
3. Accédez à Appareils.
4. Enregistrez l'appareil dans votre compte développeur Apple. Pour obtenir le nom et l'UDID de l'appareil, utilisez `ListDeviceInstances` l'API Device Farm.
5. Accédez à votre profil de mise en service et choisissez Edit (Modifier).
6. Sélectionnez l'appareil dans la liste.
7. Dans XCode, récupérez votre profil de mise en service mis à jour, puis resignez l'application.

Aucune autre configuration n'est requise. Vous pouvez maintenant créer une session d'accès à distance ou une exécution de test, et sélectionner Skip app re-signing (Ignorer la resignature d'application).

Création d'une session d'accès à distance pour faire confiance à votre application iOS

Si vous utilisez un profil de mise en service de développeur interne (entreprise), vous devez exécuter une procédure unique pour approuver le certificat du développeur d'applications interne sur chacun de vos appareils privés.

Pour ce faire, vous devez installer une application fictive signée avec le même certificat que l'application que vous souhaitez tester. Une fois que l'appareil fait confiance au profil de configuration ou au développeur d'applications d'entreprise, toutes les applications de ce développeur sont approuvées sur l'appareil privé jusqu'à ce que vous les supprimiez. Ainsi, lorsque vous installez de nouvelles versions de l'application que vous souhaitez tester, vous n'aurez pas à faire confiance à nouveau au développeur de l'application à chaque fois. Cela est particulièrement utile si vous exécutez des automatisations de test et que vous ne souhaitez pas créer une session d'accès à distance chaque fois que vous testez votre application.

Une procédure courante utilisée par de nombreux clients consiste à signer à nouveau l'[exemple d'application Device Farm pour iOS](#), puis à l'installer sur leur appareil en tant qu'application fictive.

Avant de démarrer votre session d'accès à distance, suivez les étapes décrites [Création d'un profil d'instance dans AWS Device Farm](#) pour créer ou modifier un profil d'instance dans Device Farm. Dans le profil d'instance, ajoutez l'ID de bundle de l'application fictive au paramètre Exclure les packages du nettoyage. Attachez ensuite le profil d'instance à l'instance d'appareil privée pour vous assurer que Device Farm ne supprime pas cette application de l'appareil avant qu'il ne commence un nouveau test. Cela permet de garantir que votre certificat de développeur reste approuvé.

Vous pouvez télécharger l'application fictive sur l'appareil à l'aide d'une session d'accès à distance, ce qui vous permet de lancer l'application et de faire confiance au développeur.

1. Suivez les instructions de [Création d'une session](#) pour créer une session d'accès à distance à l'aide du profil d'instance d'appareil privé que vous venez de créer. Lorsque vous créez votre session, veillez à sélectionner Skip app re-signing (Ignorer la resignature d'application).

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

Pour filtrer la liste des appareils afin d'inclure uniquement les appareils privés, sélectionnez Private device instances only (Instances d'appareils privés uniquement) pour vous assurer que vous utilisez un appareil privé avec le profil d'instance correct.

- Veillez également à ajouter l'application fictive ou l'application que vous souhaitez tester au paramètre Exclure les packages du nettoyage pour le profil d'instance attaché à cette instance.
2. Lorsque votre session à distance démarre, choisissez Choose File pour installer une application qui utilise votre profil de provisionnement interne.
 3. Lancez l'application que vous venez de charger.
 4. Vérifiez qu'une boîte de dialogue iOS apparaît indiquant que le développeur de l'application d'entreprise n'est pas fiable.
 5. Ensuite, si l'appareil iOS est doté de la version iOS 18 ou supérieure, ouvrez un ticket d'assistance auprès de l'équipe AWS Device Farm pour que notre équipe fasse confiance à l'application pour vous, car ces appareils nécessitent une approbation manuelle de l'application. Sinon, si la version iOS est 17 ou inférieure, vous pouvez accéder à l'application Paramètres et, sous Paramètres généraux, faites-vous confiance à l'application depuis le menu VPN et Profils.

Toutes les applications de ce profil de configuration ou du développeur d'applications d'entreprise sont maintenant approuvées sur cet appareil privé jusqu'à ce que vous les supprimiez.

Amazon VPC dans toutes les AWS régions d'AWS Device Farm

Les services Device Farm sont situés uniquement dans la région ouest des États-Unis (Oregonus-west-2) (). Vous pouvez utiliser Amazon Virtual Private Cloud (Amazon VPC) pour accéder à un service de votre Amazon Virtual Private Cloud dans une autre AWS région à l'aide de Device Farm. Si Device Farm et votre service se trouvent dans la même région, consultez [Utilisation des services de point de terminaison Amazon VPC avec Device Farm - Legacy \(non recommandé\)](#).

Il existe deux manières d'accéder à vos services privés situés dans une région différente. Si vos services sont situés dans une autre région qui ne l'est pas us-west-2, vous pouvez utiliser le VPC Peering pour associer le VPC de cette région à un autre VPC interfaçant avec Device Farm in. us-west-2 Toutefois, si vous avez des services dans plusieurs régions, un Transit Gateway vous permettra d'accéder à ces services avec une configuration réseau plus simple.

Pour plus d'informations, consultez les [scénarios de peering VPC](#) dans le guide d'appairage Amazon VPC.

Présentation du peering VPC VPCs dans différentes régions d'AWS Device Farm

Vous pouvez en comparer deux VPCs dans des régions différentes, à condition qu'elles comportent des blocs d'adresse CIDR distincts et ne se chevauchant pas. Cela garantit que toutes les adresses IP privées sont uniques et permet à toutes les ressources qu'elles contiennent de s'adresser les unes VPCs aux autres sans avoir besoin de toute forme de traduction d'adresses réseau (NAT). Pour plus d'informations sur la notation de bloc d'adresse CIDR, consultez [RFC 4632](#).

Cette rubrique inclut un exemple de scénario interrégional dans lequel Device Farm (appelé VPC-1) est situé dans la région de l'ouest des États-Unis (Oregon) (us-west-2). Dans cet exemple, le deuxième VPC (appelé VPC-2) se trouve dans une autre région.

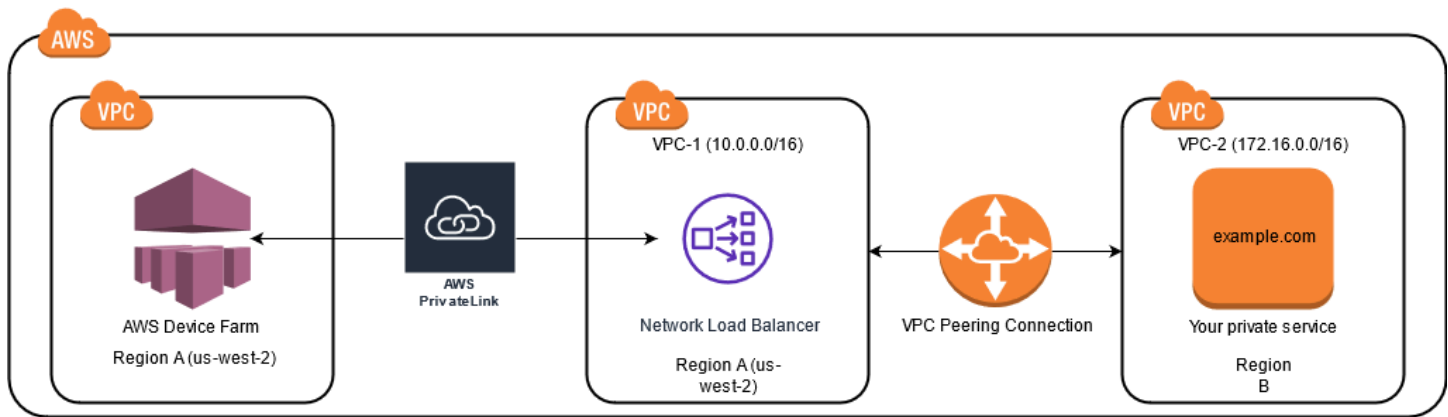
Exemple de VPC entre plusieurs régions de Device Farm

Composant du VPC	VPC-1	VPC-2
CIDR	10.0.0.0/16	172.16.0.0/16

Important

L'établissement d'une connexion de peering entre deux VPCs peut modifier la posture de sécurité du VPCs. En outre, l'ajout de nouvelles entrées à leurs tables de routage peut modifier le niveau de sécurité des ressources au sein du VPCs. Il est de votre responsabilité de mettre en œuvre ces configurations de manière à répondre aux exigences de sécurité de votre organisation. Pour plus d'informations, veuillez consulter le [modèle de responsabilité partagée](#).

Le schéma suivant montre les composants de l'exemple et les interactions entre ces composants.



Rubriques

- [Conditions préalables à l'utilisation d'Amazon VPC dans AWS Device Farm](#)
- [Étape 1 : Configuration d'une connexion de peering entre VPC-1 et VPC-2](#)
- [Étape 2 : mise à jour des tables de routage dans VPC-1 et VPC-2](#)
- [Étape 3 : Création d'un groupe cible](#)
- [Étape 4 : Création d'un Network Load Balancer](#)
- [Étape 5 : Création d'un service de point de terminaison VPC pour connecter votre VPC à Device Farm](#)
- [Étape 6 : Création d'une configuration de point de terminaison VPC entre votre VPC et Device Farm](#)
- [Étape 7 : Création d'un test pour utiliser la configuration du point de terminaison VPC](#)
- [Création d'un réseau évolutif avec Transit Gateway](#)

Conditions préalables à l'utilisation d'Amazon VPC dans AWS Device Farm

Cet exemple nécessite les éléments suivants :

- Deux VPCs qui sont configurés avec des sous-réseaux contenant des blocs CIDR ne se chevauchant pas.
- VPC-1 doit se trouver dans la us-west-2 région et contenir des sous-réseaux pour les zones de disponibilité us-west-2a, us-west-2b et us-west-2c

Pour plus d'informations sur la création VPCs et la configuration de sous-réseaux, consultez la section [Utilisation des sous-réseaux VPCs et sous-réseaux](#) dans le manuel Amazon VPC Peering Guide.

Étape 1 : Configuration d'une connexion de peering entre VPC-1 et VPC-2

Établissez une connexion d'appairage entre les deux VPCs contenant des blocs CIDR ne se chevauchant pas. Pour ce faire, consultez la section [Créer et accepter des connexions d'appairage VPC dans le guide d'appairage](#) Amazon VPC. À l'aide du scénario interrégional de cette rubrique et du guide d'appairage Amazon VPC, l'exemple de configuration de connexion d'appairage suivant est créé :

Nom

Device-Farm-Peering-Connection-1

ID VPC (demandeur)

vpc-0987654321gfedcba (VPC-2)

Compte

My account

Région

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

ID VPC (Accepteur)

vpc-1234567890abcdefg (VPC-1)

Note

Assurez-vous de consulter vos quotas de connexion d'appairage VPC lorsque vous établissez de nouvelles connexions d'appairage. Pour plus d'informations, consultez les [quotas Amazon VPC](#) dans le guide de peering Amazon VPC.

Étape 2 : mise à jour des tables de routage dans VPC-1 et VPC-2

Après avoir configuré une connexion d'appairage, vous devez établir un itinéraire de destination entre les deux VPCs pour transférer des données entre eux. Pour établir cette route, vous pouvez

mettre à jour manuellement la table de routage de VPC-1 afin qu'elle pointe vers le sous-réseau de VPC-2 et vice versa. Pour ce faire, consultez [Mettre à jour vos tables de routage pour une connexion d'appairage VPC dans le guide d'appairage](#) Amazon VPC. À l'aide du scénario interrégional de cette rubrique et du Amazon VPC Peering Guide, l'exemple de configuration de table de routage suivant est créé :

Exemple de table de routage VPC Device Farm

Composant VPC	VPC-1	VPC-2
ID de table de routage	rtb-1234567890abcdefg	rtb-0987654321gfedcba
Plage d'adresses locales	10.0.0.0/16	172.16.0.0/16
Plage d'adresses de destination	172.16.0.0/16	10.0.0.0/16

Étape 3 : Création d'un groupe cible

Après avoir configuré vos itinéraires de destination, vous pouvez configurer un Network Load Balancer dans VPC-1 pour acheminer les demandes vers VPC-2.

Le Network Load Balancer doit d'abord contenir un groupe cible contenant les adresses IP auxquelles les demandes sont envoyées.

Pour créer un groupe cible

1. Identifiez les adresses IP du service que vous souhaitez cibler dans VPC-2.
 - Ces adresses IP doivent être membres du sous-réseau utilisé dans la connexion d'appairage.
 - Les adresses IP ciblées doivent être statiques et immuables. Si votre service possède des adresses IP dynamiques, envisagez de cibler une ressource statique (telle qu'un Network Load Balancer) et de faire en sorte que cette ressource statique achemine les demandes vers votre véritable cible.

Note

- Si vous ciblez une ou plusieurs instances autonomes Amazon Elastic Compute Cloud (Amazon EC2), ouvrez la console Amazon EC2 à l'adresse, puis choisissez Instances.
<https://console.aws.amazon.com/ec2/>

- Si vous ciblez un groupe Amazon EC2 Auto Scaling d'instances Amazon EC2, vous devez associer le groupe Amazon EC2 Auto Scaling à un Network Load Balancer. Pour plus d'informations, veuillez consulter [Attacher un équilibreur de charge à votre groupe Auto Scaling](#) (langue française non garantie) dans le Guide de l'utilisateur Amazon EC2 Auto Scaling.

Vous pouvez ensuite ouvrir la console Amazon EC2 à l'adresse <https://console.aws.amazon.com/ec2/>, puis choisir Network Interfaces. À partir de là, vous pouvez consulter les adresses IP de chacune des interfaces réseau du Network Load Balancer dans chaque zone de disponibilité.

2. Créez un groupe cible dans VPC-1. Pour ce faire, reportez-vous à la section [Création d'un groupe cible pour votre Network Load Balancer](#) dans le Guide de l'utilisateur pour les Network Load Balancers.

Les groupes cibles pour les services d'un autre VPC nécessitent la configuration suivante :

- Pour Choisir un type de cible, choisissez les adresses IP.
- Pour le VPC, choisissez le VPC qui hébergera l'équilibreur de charge. Pour l'exemple du sujet, il s'agira de VPC-1.
- Sur la page Enregistrer les cibles, enregistrez une cible pour chaque adresse IP dans VPC-2.

Pour Réseau, sélectionnez Autre adresse IP privée.

Pour la zone de disponibilité, choisissez les zones souhaitées dans VPC-1.

Pour IPv4 l'adresse, choisissez l'adresse IP VPC-2.

Dans Ports, choisissez vos ports.

- Choisissez Inclure comme en attente ci-dessous. Lorsque vous avez fini de spécifier les adresses, choisissez Enregistrer les cibles en attente.

À l'aide du scénario interrégional de cette rubrique et du guide de l'utilisateur pour les équilibreurs de charge réseau, les valeurs suivantes sont utilisées dans la configuration du groupe cible :

Target type (Type de cible)

IP addresses

Nom du groupe cible

`my-target-group`

Protocole/Port

TCP : 80

VPC

`vpc-1234567890abcdefg` (VPC-1)

Réseau

Other private IP address

Zone de disponibilité

`all`

IPv4 address

`172.16.100.60`

Ports

80

Étape 4 : Création d'un Network Load Balancer

Créez un Network Load Balancer en utilisant le groupe cible décrit à l'[étape](#) 3. Pour ce faire, consultez [Creating a Network Load Balancer](#).

À l'aide du scénario interrégional de cette rubrique, les valeurs suivantes sont utilisées dans un exemple de configuration Network Load Balancer :

Nom de l'équilibreur de charge

`my-nlb`

Schéma

`Internal`

VPC

`vpc-1234567890abcdefg` (VPC-1)

Mappage

us-west-2a - subnet-4i23iuufkdiuflsloi

us-west-2b - subnet-7x989pkjj78nmn23j

us-west-2c - subnet-0231ndmas12bnnsds

Protocole/Port

TCP : 80

Groupe cible

my-target-group

Étape 5 : Création d'un service de point de terminaison VPC pour connecter votre VPC à Device Farm

Vous pouvez utiliser le Network Load Balancer pour créer un service de point de terminaison VPC. Grâce à ce service de point de terminaison VPC, Device Farm peut se connecter à votre service dans VPC-2 sans aucune infrastructure supplémentaire, telle qu'une passerelle Internet, une instance NAT ou une connexion VPN.

Pour ce faire, consultez [Création d'un service de point de terminaison Amazon VPC](#).

Étape 6 : Création d'une configuration de point de terminaison VPC entre votre VPC et Device Farm

Vous pouvez désormais établir une connexion privée entre votre VPC et Device Farm. Vous pouvez utiliser Device Farm pour tester des services privés sans les exposer via l'Internet public. Pour ce faire, consultez [Création d'une configuration de point de terminaison VPC dans Device Farm](#).

À l'aide du scénario interrégional de cette rubrique, les valeurs suivantes sont utilisées dans un exemple de configuration de point de terminaison VPC :

Nom

My VPCE Configuration

Nom du service VPCE

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

Nom DNS du service

devicefarm.com

Étape 7 : Création d'un test pour utiliser la configuration du point de terminaison VPC

Vous pouvez créer des tests qui utilisent la configuration du point de terminaison VPC décrite à l'[étape 6](#). Pour plus d'informations, consultez [Création d'un test dans Device Farm](#) ou [Création d'une session](#).

Création d'un réseau évolutif avec Transit Gateway

Pour créer un réseau évolutif en utilisant plus de deux VPCs, vous pouvez utiliser Transit Gateway pour faire office de hub de transit afin d'interconnecter votre réseau VPCs et vos réseaux locaux. Pour configurer un VPC dans la même région que Device Farm afin d'utiliser un Transit Gateway, vous pouvez suivre le guide Amazon [VPC Endpoint Services with Device Farm](#) pour cibler les ressources d'une autre région en fonction de leurs adresses IP privées.

Pour plus d'informations sur Transit Gateway, voir [Qu'est-ce qu'une passerelle de transit ?](#) dans le guide Amazon VPC Transit Gateways.

Résiliation d'appareils privés dans Device Farm

Pour résilier un appareil privé après la durée initiale convenue, vous devez fournir un préavis de 30 jours de non-renouvellement par e-mail à l'adresse <aws-devicefarm-support@amazon> .com. Pour plus d'informations sur les appareils privés, consultez [Appareils privés dans AWS Device Farm](#).

Important

Ces instructions s'appliquent uniquement à la résiliation des contrats d'appareils privés. Pour tous les autres AWS services et problèmes de facturation, consultez la documentation correspondante de ces produits ou contactez le AWS support.

VPC-ENI dans AWS Device Farm

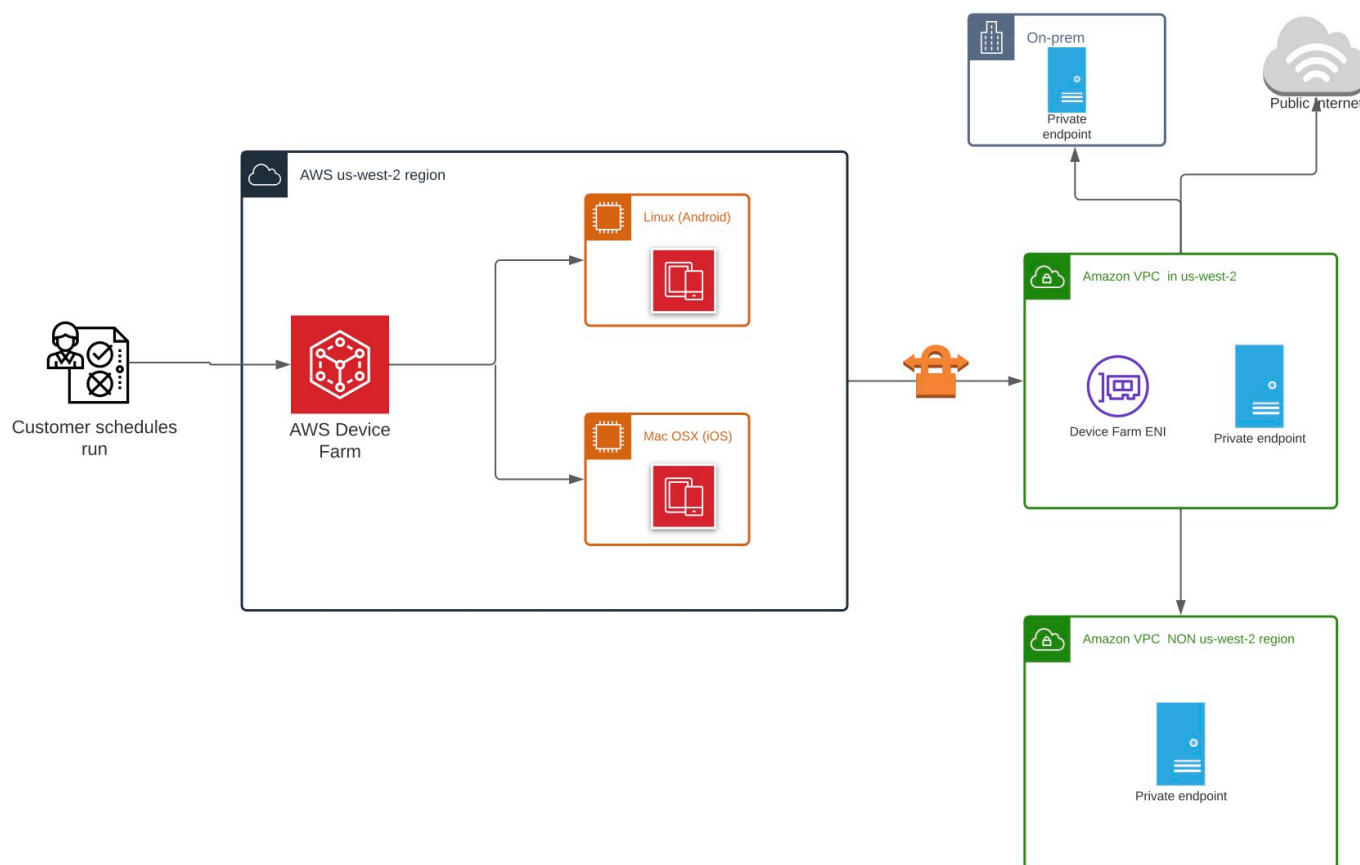
Warning

Cette fonctionnalité n'est disponible que sur les [appareils privés](#). Pour demander l'utilisation d'un appareil privé sur votre AWS compte, veuillez [nous contacter](#). Si des appareils privés ont déjà été ajoutés à votre AWS compte, nous vous recommandons vivement d'utiliser cette méthode de connectivité VPC.

La fonctionnalité de connectivité VPC-ENI d'AWS Device Farm aide les clients à se connecter en toute sécurité à leurs points de terminaison privés hébergés sur un logiciel sur AWS site ou sur un autre fournisseur de cloud.

Vous pouvez connecter à la fois les appareils mobiles Device Farm et leurs machines hôtes à un environnement Amazon Virtual Private Cloud (Amazon VPC) de la us-west-2 région, qui permet d'accéder à des non-internet-facing services et applications isolés via une interface [elastic network](#). Pour plus d'informations VPCs, consultez le guide de l'[utilisateur Amazon VPC](#).

Si votre point de terminaison privé ou VPC ne se trouve pas dans la us-west-2 région, vous pouvez le relier à un VPC de la us-west-2 région à l'aide de solutions telles qu'un Transit Gateway ou un VPC peering. Dans de telles situations, Device Farm créera une ENI dans un sous-réseau que vous fournissez pour votre VPC de us-west-2 région, et vous serez chargé de veiller à ce qu'une connexion puisse être établie entre le VPC de us-west-2 région et le VPC de l'autre région.



Pour plus d'informations sur l'utilisation AWS CloudFormation pour créer et comparer automatiquement VPCs, consultez les [VPCPeering modèles](#) dans le référentiel de AWS CloudFormation modèles sur GitHub.

Note

Device Farm ne facture rien pour la création ENIs dans le VPC d'un client dans. us-west-2. Le coût de la connectivité interrégionale ou externe entre VPC n'est pas inclus dans cette fonctionnalité.

Une fois que vous avez configuré l'accès au VPC, les appareils et les machines hôtes que vous utilisez pour vos tests ne pourront pas se connecter à des ressources extérieures au VPC (par

exemple, publiques CDNs) sauf s'il existe une passerelle NAT que vous spécifiez au sein du VPC. Pour plus d'informations, veuillez consulter [NAT Gateways \(Passerelles NAT\)](#) dans le Guide de l'utilisateur Amazon VPC.

Rubriques

- [AWS contrôle d'accès et IAM](#)
- [Rôles liés à un service](#)
- [Conditions préalables](#)
- [Connexion à Amazon VPC](#)
- [Restrictions](#)
- [Utilisation des services de point de terminaison Amazon VPC avec Device Farm - Legacy \(non recommandé\)](#)

AWS contrôle d'accès et IAM

AWS Device Farm vous permet d'utiliser [Gestion des identités et des accès AWS](#)(IAM) pour créer des politiques accordant ou restreignant l'accès aux fonctionnalités de Device Farm. Pour utiliser la fonctionnalité de connectivité VPC avec AWS Device Farm, la politique IAM suivante est requise pour le compte utilisateur ou le rôle que vous utilisez pour accéder à AWS Device Farm :

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "devicefarm:*",
      "ec2:DescribeVpcs",
      "ec2:DescribeSubnets",
      "ec2:DescribeSecurityGroups",
      "ec2:CreateNetworkInterface"
    ],
    "Resource": [
      "*"
    ]
  }],
}
```

```
{
  "Effect": "Allow",
  "Action": "iam:CreateServiceLinkedRole",
  "Resource": "arn:aws:iam::*:role/aws-service-role/devicefarm.amazonaws.com/AWSServiceRoleForDeviceFarm",
  "Condition": {
    "StringLike": {
      "iam:AWSServiceName": "devicefarm.amazonaws.com"
    }
  }
}
```

Pour créer ou mettre à jour un projet Device Farm avec une configuration VPC, votre politique IAM doit vous permettre d'effectuer les actions suivantes sur les ressources répertoriées dans la configuration VPC :

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

En outre, votre politique IAM doit également autoriser la création du rôle lié au service :

```
"iam:CreateServiceLinkedRole"
```

Note

Aucune de ces autorisations n'est requise pour les utilisateurs qui n'utilisent pas de configurations VPC dans leurs projets.

Rôles liés à un service

AWS Device Farm utilise des rôles Gestion des identités et des accès AWS liés à un [service](#) (IAM). Un rôle lié à un service est un type unique de rôle IAM directement lié à Device Farm. Les rôles liés à un service sont prédéfinis par Device Farm et incluent toutes les autorisations dont le service a besoin pour appeler d'autres AWS services en votre nom.

Un rôle lié à un service facilite la configuration de Device Farm, car vous n'avez pas à ajouter manuellement les autorisations nécessaires. Device Farm définit les autorisations associées à ses rôles liés aux services et, sauf indication contraire, seul Device Farm peut assumer ses rôles. Les autorisations définies comprennent la politique de confiance et la politique d'autorisation. De plus, cette politique d'autorisation ne peut pas être attachée à une autre entité IAM.

Vous pouvez supprimer un rôle lié à un service uniquement après la suppression préalable de ses ressources connexes. Cela protège les ressources de votre Device Farm, car vous ne pouvez pas supprimer par inadvertance l'autorisation d'accès aux ressources.

Pour de plus amples informations sur les autres services qui prennent en charge les rôles liés à un service, veuillez consulter [Services AWS qui fonctionnent avec IAM](#) et rechercher les services qui ont Yes (Oui) dans la colonne Service-Linked Role (Rôle lié à un service). Sélectionnez un Yes (Oui) avec un lien permettant de consulter la documentation du rôle lié à un service, pour ce service.

Autorisations de rôle liées à un service pour Device Farm

Device Farm utilise le rôle lié à un service nommé `AWSServiceRoleForDeviceFarm`— Autorise Device Farm à accéder aux ressources AWS en votre nom.

Le rôle `AWSService RoleForDeviceFarm` lié à un service fait confiance aux services suivants pour assumer le rôle :

- `devicefarm.amazonaws.com`

La politique d'autorisation des rôles permet à Device Farm d'effectuer les actions suivantes :

- Pour votre compte
 - Création d'interfaces réseau
 - Décrire les interfaces réseau
 - Décrivez VPCs
 - Décrire les sous-réseaux
 - Décrire des groupes de sécurité
 - Supprimer des interfaces
 - Modifier les interfaces réseau
- Pour les interfaces réseau
 - Création de tags

- Pour les interfaces réseau EC2 gérées par Device Farm
 - Création d'autorisations d'interface réseau

La politique IAM complète se lit comme suit :

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeVpcs",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkInterface"
      ],
      "Resource": [
        "arn:aws:ec2:*:*:subnet/*",
        "arn:aws:ec2:*:*:security-group/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkInterface"
      ],
      "Resource": [
        "arn:aws:ec2:*:*:network-interface/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/AWSDeviceFarmManaged": "true"
        }
      }
    }
  ]
}
```



```
    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "ec2:CreateTags"
    ],
    "Resource": "arn:aws:ec2:*:*:network-interface/*",
    "Condition": {
      "StringEquals": {
        "ec2:CreateAction": "CreateNetworkInterface"
      }
    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "ec2:CreateNetworkInterfacePermission",
      "ec2>DeleteNetworkInterface"
    ],
    "Resource": "arn:aws:ec2:*:*:network-interface/*",
    "Condition": {
      "StringEquals": {
        "aws:ResourceTag/AWSDeviceFarmManaged": "true"
      }
    }
  },
  {
    "Effect": "Allow",
    "Action": [
      "ec2:ModifyNetworkInterfaceAttribute"
    ],
    "Resource": [
      "arn:aws:ec2:*:*:security-group/*",
      "arn:aws:ec2:*:*:instance/*"
    ]
  },
  {
    "Effect": "Allow",
    "Action": [
      "ec2:ModifyNetworkInterfaceAttribute"
    ],
    "Resource": "arn:aws:ec2:*:*:network-interface/*",
    "Condition": {
```

```
"StringEquals": {  
  "aws:ResourceTag/AWSDeviceFarmManaged": "true"  
}  
}  
}  
]  
}
```

Vous devez configurer les autorisations de manière à permettre à une entité IAM (comme un utilisateur, un groupe ou un rôle) de créer, modifier ou supprimer un rôle lié à un service. Pour en savoir plus, consultez [Service-Linked Role Permissions \(autorisations du rôle lié à un service\)](#) dans le Guide de l'utilisateur IAM.

Création d'un rôle lié à un service pour Device Farm

Lorsque vous fournissez une configuration VPC pour un projet de test mobile, il n'est pas nécessaire de créer manuellement un rôle lié à un service. Lorsque vous créez votre première ressource Device Farm dans l' AWS Management Console AWS API AWS CLI, Device Farm crée le rôle lié au service pour vous.

Si vous supprimez ce rôle lié à un service et que vous avez ensuite besoin de le recréer, vous pouvez utiliser la même procédure pour recréer le rôle dans votre compte. Lorsque vous créez votre première ressource Device Farm, Device Farm crée à nouveau le rôle lié au service pour vous.

Vous pouvez également utiliser la console IAM pour créer un rôle lié à un service avec le cas d'utilisation de Device Farm. Dans l'API AWS CLI ou dans l' AWS API, créez un rôle lié à un service avec le nom du `devicefarm.amazonaws.com` service. Pour plus d'informations, consultez [Création d'un rôle lié à un service](#) dans le Guide de l'utilisateur IAM. Si vous supprimez ce rôle lié à un service, vous pouvez utiliser ce même processus pour créer le rôle à nouveau.

Modification d'un rôle lié à un service pour Device Farm

Device Farm ne vous permet pas de modifier le rôle `AWSService RoleForDeviceFarm` lié au service. Après avoir créé un rôle lié à un service, vous ne pouvez pas changer le nom du rôle, car plusieurs entités peuvent faire référence à ce rôle. Néanmoins, vous pouvez modifier la description du rôle à l'aide d'IAM. Pour en savoir plus, consultez [Modification d'un rôle lié à un service](#) dans le Guide de l'utilisateur IAM.

Supprimer un rôle lié à un service pour Device Farm

Si vous n'avez plus besoin d'utiliser une fonctionnalité ou un service qui nécessite un rôle lié à un service, nous vous recommandons de supprimer ce rôle. De cette façon, vous n'avez aucune entité inutilisée qui n'est pas surveillée ou gérée activement. Cependant, vous devez nettoyer les ressources de votre rôle lié à un service avant de pouvoir les supprimer manuellement.

Note

Si le service Device Farm utilise le rôle lorsque vous essayez de supprimer les ressources, la suppression risque d'échouer. Si cela se produit, patientez quelques minutes et réessayez.

Pour supprimer manuellement le rôle lié au service à l'aide d'IAM

Utilisez la console IAM, le AWS CLI, ou l' AWS API pour supprimer le rôle lié au AWSService RoleForDeviceFarm service. Pour en savoir plus, consultez [Suppression d'un rôle lié à un service](#) dans le Guide de l'utilisateur IAM.

Régions prises en charge pour les rôles liés au service Device Farm

Device Farm prend en charge l'utilisation de rôles liés à un service dans toutes les régions où le service est disponible. Pour plus d'informations, consultez [AWS Régions et points de terminaison](#).

Device Farm ne prend pas en charge l'utilisation de rôles liés à un service dans toutes les régions où le service est disponible. Vous pouvez utiliser le AWSService RoleForDeviceFarm rôle dans les régions suivantes.

Nom de la région	Identité de la région	Support dans Device Farm
USA Est (Virginie du Nord)	us-east-1	Non
USA Est (Ohio)	us-east-2	Non
USA Ouest (Californie du Nord)	us-west-1	Non
USA Ouest (Oregon)	us-west-2	Oui
Asie-Pacifique (Mumbai)	ap-south-1	Non

Nom de la région	Identité de la région	Support dans Device Farm
Asie-Pacifique (Osaka)	ap-northeast-3	Non
Asie-Pacifique (Séoul)	ap-northeast-2	Non
Asie-Pacifique (Singapour)	ap-southeast-1	Non
Asie-Pacifique (Sydney)	ap-southeast-2	Non
Asie-Pacifique (Tokyo)	ap-northeast-1	Non
Canada (Centre)	ca-central-1	Non
Europe (Francfort)	eu-central-1	Non
Europe (Irlande)	eu-west-1	Non
Europe (Londres)	eu-west-2	Non
Europe (Paris)	eu-west-3	Non
Amérique du Sud (São Paulo)	sa-east-1	Non
AWS GovCloud (US)	us-gov-west-1	Non

Conditions préalables

La liste suivante décrit certaines exigences et suggestions à prendre en compte lors de la création de configurations VPC-ENI :

- Les appareils privés doivent être attribués à votre AWS compte.
- Vous devez disposer d'un AWS compte, d'un utilisateur ou d'un rôle autorisé pour créer un rôle lié à un service. Lorsque vous utilisez des points de terminaison Amazon VPC dotés des fonctionnalités de test mobile de Device Farm, Device Farm crée un rôle lié à un service Gestion des identités et des accès AWS (IAM).
- Device Farm ne peut se connecter VPCs qu'à la us-west-2 Région. Si vous n'avez pas de VPC dans la us-west-2 région, vous devez en créer un. Ensuite, pour accéder aux ressources d'un

VPC d'une autre région, vous devez établir une connexion d'appairage entre le VPC de la région `us-west-2` et le VPC de l'autre région. Pour plus d'informations sur le peering VPCs, consultez le guide de peering [Amazon VPC](#).

Vous devez vérifier que vous avez accès au VPC que vous avez spécifié lorsque vous configurez la connexion. Vous devez configurer certaines autorisations Amazon Elastic Compute Cloud (Amazon EC2) pour Device Farm.

- La résolution DNS est requise dans le VPC que vous utilisez.
- Une fois votre VPC créé, vous aurez besoin des informations suivantes sur le VPC de la région : `us-west-2`
 - ID du VPC
 - Sous-réseau IDs (sous-réseaux privés uniquement)
 - Groupe de sécurité IDs
- Vous devez configurer les connexions Amazon VPC par projet. Pour le moment, vous ne pouvez configurer qu'une seule configuration VPC par projet. Lorsque vous configurez un VPC, Amazon VPC crée une interface au sein de votre VPC et l'affecte aux sous-réseaux et groupes de sécurité spécifiés. Toutes les futures sessions associées au projet utiliseront la connexion VPC configurée.
- Vous ne pouvez pas utiliser les configurations VPC-ENI avec l'ancienne fonctionnalité VPCE.
- Nous vous recommandons vivement de ne pas mettre à jour un projet existant avec une configuration VPC-ENI, car les projets existants peuvent avoir des paramètres VPCE qui persistent au niveau de l'exécution. Au lieu de cela, si vous utilisez déjà les fonctionnalités VPCE existantes, utilisez VPC-ENI pour tous les nouveaux projets.

Connexion à Amazon VPC

Vous pouvez configurer et mettre à jour votre projet pour utiliser les points de terminaison Amazon VPC. La configuration VPC-ENI est configurée par projet. Un projet ne peut avoir qu'un seul point de terminaison VPC-ENI à la fois. Pour configurer l'accès VPC pour un projet, vous devez connaître les informations suivantes :

- L'ID VPC indique `us-west-2` si votre application y est hébergée ou l'ID `us-west-2` VPC qui se connecte à un autre VPC dans une autre région.
- Les groupes de sécurité applicables à appliquer à la connexion.
- Les sous-réseaux qui seront associés à la connexion. Lorsqu'une session démarre, le plus grand sous-réseau disponible est utilisé. Nous vous recommandons d'associer plusieurs sous-réseaux

à différentes zones de disponibilité afin d'améliorer le niveau de disponibilité de votre connectivité VPC.

- Lors de l'utilisation de VPC-ENI, le résolveur DNS utilisé par les hôtes et appareils de test Device Farm sera le serveur fourni par les services DHCP dans le sous-réseau du client. Dans une configuration par défaut, il s'agira du résolveur par défaut du VPC. Les clients qui souhaitent spécifier des résolveurs DNS personnalisés peuvent configurer un ensemble d'options DHCP dans leur VPC.

Une fois que vous avez créé votre configuration VPC-ENI, vous pouvez mettre à jour ses détails à l'aide de la console ou de la CLI en suivant les étapes ci-dessous.

Console

1. Connectez-vous à la console Device Farm à l'adresse <https://console.aws.amazon.com/devicefarm>.
2. Sur le panneau de navigation de Device Farm, choisissez Mobile Device Testing, puis Projects.
3. Sous Projets de test mobile, choisissez le nom de votre projet dans la liste.
4. Choisissez Project settings (Paramètres du projet).
5. Dans la section Paramètres du Virtual Private Cloud (VPC), vous pouvez modifier les paramètres VPC, Subnets (sous-réseaux privés uniquement) et Security Groups.
6. Choisissez Enregistrer.

CLI

Utilisez la commande AWS CLI suivante pour mettre à jour l'Amazon VPC :

```
$ aws devicefarm update-project \
--arn arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef \
--vpc-config \
securityGroupIds=sg-02c1537701a7e3763,sg-005dadf9311efda25,\
subnetIds=subnet-09b1a45f9cac53717,subnet-09b1a45f9cac12345,\
vpcId=vpc-0238fb322af81a368
```

Vous pouvez également configurer un Amazon VPC lors de la création de votre projet :

```
$ aws devicefarm create-project \
--name VPCDemo \
--vpc-config \
securityGroupIds=sg-02c1537701a7e3763,sg-005dadf9311efda25,\
subnetIds=subnet-09b1a45f9cac53717,subnet-09b1a45f9cac12345,\
vpcId=vpc-0238fb322af81a368
```

Restrictions

Les limitations suivantes s'appliquent à la fonctionnalité VPC-ENI :

- Vous pouvez fournir jusqu'à cinq groupes de sécurité dans la configuration VPC d'un projet Device Farm.
- Vous pouvez fournir jusqu'à huit sous-réseaux dans la configuration VPC d'un projet Device Farm.
- Lorsque vous configurez un projet Device Farm pour qu'il fonctionne avec votre VPC, le plus petit sous-réseau que vous pouvez fournir doit avoir au moins cinq adresses disponibles. IPv4
- Les adresses IP publiques ne sont pas prises en charge pour le moment. Nous vous recommandons plutôt d'utiliser des sous-réseaux privés dans vos projets Device Farm. Si vous avez besoin d'un accès public à Internet pendant vos tests, utilisez une [passerelle de traduction d'adresses réseau \(NAT\)](#). La configuration d'un projet Device Farm avec un sous-réseau public ne donne pas à vos tests un accès Internet ou une adresse IP publique.
- L'intégration VPC-ENI ne prend en charge que les sous-réseaux privés de votre VPC.
- Seul le trafic sortant de l'ENI géré par le service est pris en charge. Cela signifie que l'ENI ne peut pas recevoir de demandes entrantes non sollicitées de la part du VPC.

Utilisation des services de point de terminaison Amazon VPC avec Device Farm - Legacy (non recommandé)

Warning

Nous vous recommandons vivement d'utiliser la connectivité VPC-ENI décrite sur [cette](#) page pour la connectivité des terminaux privés, car le VPCE est désormais considéré comme une fonctionnalité héritée. Le VPC-ENI offre plus de flexibilité, des configurations plus simples, est

plus rentable et nécessite beaucoup moins de frais de maintenance par rapport à la méthode de connectivité VPCE.

 Note

L'utilisation d'Amazon VPC Endpoint Services avec Device Farm n'est prise en charge que pour les clients disposant d'appareils privés configurés. Pour permettre à votre compte AWS d'utiliser cette fonctionnalité avec des appareils privés, veuillez [nous contacter](#).

Amazon Virtual Private Cloud (Amazon VPC) est un AWS service que vous pouvez utiliser pour lancer AWS des ressources dans un réseau virtuel que vous définissez. Avec un VPC, vous pouvez contrôler vos paramètres réseau, tels que la plage d'adresses IP, les sous-réseaux, les tables de routage et les passerelles réseau.

Si vous utilisez Amazon VPC pour héberger des applications privées dans la AWS région USA Ouest (Oregon) (us-west-2), vous pouvez établir une connexion privée entre votre VPC et Device Farm. Grâce à cette connexion, vous pouvez utiliser Device Farm pour tester des applications privées sans les exposer via l'Internet public. Pour permettre à votre AWS compte d'utiliser cette fonctionnalité avec des appareils privés, [contactez-nous](#).

Pour connecter une ressource de votre VPC à Device Farm, vous pouvez utiliser la console Amazon VPC pour créer un service de point de terminaison VPC. Ce service de point de terminaison vous permet de fournir la ressource de votre VPC à Device Farm via un point de terminaison VPC Device Farm. Le service de point de terminaison fournit une connectivité fiable et évolutive à Device Farm sans nécessiter de passerelle Internet, d'instance de traduction d'adresses réseau (NAT) ou de connexion VPN. Pour plus d'informations, consultez la section [Services de point de terminaison VPC PrivateLink \(AWS\)](#) dans le AWS PrivateLink Guide.

 Important

La fonctionnalité de point de terminaison VPC de Device Farm vous permet de connecter en toute sécurité les services internes privés de votre VPC au VPC public Device Farm à l'aide de connexions. AWS PrivateLink Bien que la connexion soit sécurisée et privée, la sécurité dépend de votre protection de vos informations d'identification AWS . Si vos AWS

informations d'identification sont compromises, un attaquant peut accéder à vos données de service ou les exposer au monde extérieur.

Après avoir créé un service de point de terminaison VPC dans Amazon VPC, vous pouvez utiliser la console Device Farm pour créer une configuration de point de terminaison VPC dans Device Farm. Cette rubrique explique comment créer la connexion Amazon VPC et la configuration du point de terminaison VPC dans Device Farm.

Avant de commencer

Les informations suivantes sont destinées aux utilisateurs d'Amazon VPC de la région USA Ouest (Oregon) (us-west-2), avec un sous-réseau dans chacune des zones de disponibilité suivantes : us-west-2a, us-west-2b et us-west-2c.

Device Farm a des exigences supplémentaires concernant les services de point de terminaison VPC avec lesquels vous pouvez l'utiliser. Lorsque vous créez et configurez un service de point de terminaison VPC pour qu'il fonctionne avec Device Farm, assurez-vous de choisir des options qui répondent aux exigences suivantes :

- Les zones de disponibilité du service doivent inclure us-west-2a, us-west-2b et us-west-2c. Le Network Load Balancer associé à un service de point de terminaison VPC détermine les zones de disponibilité pour ce service de point de terminaison VPC. Si votre service de point de terminaison VPC n'affiche pas ces trois zones de disponibilité, vous devez recréer votre Network Load Balancer pour activer ces trois zones, puis réassocier le Network Load Balancer à votre service de point de terminaison.
- Les principaux autorisés pour le service de point de terminaison doivent inclure l'Amazon Resource Name (ARN) du point de terminaison VPC Device Farm (ARN du service). Après avoir créé votre service de point de terminaison, ajoutez l'ARN du service de point de terminaison VPC de Device Farm à votre liste d'autorisation pour autoriser Device Farm à accéder à votre service de point de terminaison VPC. Pour obtenir l'ARN du service de point de terminaison VPC Device Farm, [contactez-nous](#).

En outre, si vous maintenez le paramètre Acceptance requise activé lorsque vous créez votre service de point de terminaison VPC, vous devez accepter manuellement chaque demande de connexion envoyée par Device Farm au service de point de terminaison. Pour modifier ce paramètre pour un service de point de terminaison existant, choisissez le service de point de terminaison sur la console

Amazon VPC, choisissez Actions, puis choisissez Modifier le paramètre d'acceptation du point de terminaison. Pour plus d'informations, voir [Modifier les équilibreurs de charge et les paramètres d'acceptation](#) dans le AWS PrivateLink Guide.

La section suivante explique comment créer un service de point de terminaison Amazon VPC répondant à ces exigences.

Étape 1 : Création d'un Network Load Balancer

La première étape pour établir une connexion privée entre votre VPC et Device Farm consiste à créer un Network Load Balancer pour acheminer les demandes vers un groupe cible.

New console

Pour créer un Network Load Balancer à l'aide de la nouvelle console

1. Ouvrez la console Amazon Elastic Compute Cloud (Amazon EC2) à l'adresse. <https://console.aws.amazon.com/ec2/>
2. Dans le volet de navigation, sous Équilibrage de charge, choisissez Load balancers.
3. Choisissez Créer un équilibreur de charge.
4. Sous Network load balancer, choisissez Create.
5. Sur la page Créer un équilibreur de charge réseau, sous Configuration de base, procédez comme suit :
 - a. Entrez le nom d'un équilibreur de charge.
 - b. Pour Schéma, choisissez Internal.
6. Sous Network mapping (Mappage réseau), procédez comme suit :
 - a. Choisissez le VPC pour votre groupe cible.
 - b. Sélectionnez les mappages suivants :
 - us-west-2a
 - us-west-2b
 - us-west-2c
7. Sous Écouteurs et routage, utilisez les options Protocole et Port pour choisir votre groupe cible.

 Note

Par défaut, l'équilibrage de charge des zones de disponibilité croisée est désactivé. Étant donné que l'équilibreur de charge utilise les zones us-west-2a de disponibilité us-west-2c, il nécessite que les cibles soient enregistrées dans chacune de ces zones de disponibilité ou, si vous enregistrez des cibles dans moins de trois zones, il nécessite que vous activiez l'équilibrage de charge entre zones. us-west-2b Dans le cas contraire, l'équilibreur de charge risque de ne pas fonctionner comme prévu.

8. Choisissez Créer un équilibreur de charge.

Old console

Pour créer un Network Load Balancer à l'aide de l'ancienne console

1. Ouvrez la console Amazon Elastic Compute Cloud (Amazon EC2) à l'adresse. <https://console.aws.amazon.com/ec2/>
2. Dans le volet de navigation, sous Équilibrage de charge, choisissez les équilibreurs de charge.
3. Choisissez Créer un équilibreur de charge.
4. Sous Network load balancer, choisissez Create.
5. Sur la page Configurer l'équilibreur de charge, sous Configuration de base, procédez comme suit :
 - a. Entrez le nom d'un équilibreur de charge.
 - b. Pour Schéma, choisissez Internal.
6. Sous Écouteurs, sélectionnez le protocole et le port utilisés par votre groupe cible.
7. Sous Zones de disponibilité, procédez comme suit :
 - a. Choisissez le VPC pour votre groupe cible.
 - b. Sélectionnez les zones de disponibilité suivantes :
 - us-west-2a
 - us-west-2b

- `us-west-2c`
- c. Choisissez Suivant : configurer les paramètres de sécurité.
- 8. (Facultatif) Configurez vos paramètres de sécurité, puis choisissez Suivant : configurer le routage.
- 9. Sur la page Configure Routing, procédez de la façon suivante :
 - a. Pour Groupe cible, choisissez Groupe cible existant.
 - b. Dans Nom, choisissez votre groupe cible.
 - c. Choisissez Suivant : enregistrer les cibles.
- 10. Sur la page Enregistrer les cibles, passez en revue vos cibles, puis choisissez Suivant : révision.

 Note

Par défaut, l'équilibrage de charge des zones de disponibilité croisée est désactivé. Étant donné que l'équilibreur de charge utilise les zones `us-west-2a` de disponibilité et `us-west-2c`, il nécessite que les cibles soient enregistrées dans chacune de ces zones de disponibilité ou, si vous enregistrez des cibles dans moins de trois zones, il nécessite que vous activiez l'équilibrage de charge entre zones. `us-west-2b` Dans le cas contraire, l'équilibreur de charge risque de ne pas fonctionner comme prévu.

- 11. Vérifiez la configuration de votre équilibreur de charge, puis choisissez Create.

Étape 2 : Création d'un service de point de terminaison Amazon VPC

Après avoir créé le Network Load Balancer, utilisez la console Amazon VPC pour créer un service de point de terminaison dans votre VPC.

1. Ouvrez la console Amazon VPC à l'adresse <https://console.aws.amazon.com/vpc/>.
2. Sous Ressources par région, choisissez Endpoint services.
3. Choisissez Create endpoint service (Créer un service de point de terminaison).
4. Effectuez l'une des actions suivantes :
 - Si vous possédez déjà un Network Load Balancer que vous souhaitez que le service de point de terminaison utilise, choisissez-le sous Available load balancers, puis passez à l'étape 5.

- Si vous n'avez pas encore créé de Network Load Balancer, choisissez Create new load balancer. La console Amazon EC2 s'ouvre. Suivez les étapes décrites dans [Creating a Network Load Balancer](#) en commençant par l'étape 3, puis poursuivez ces étapes dans la console Amazon VPC.
5. Pour les zones de disponibilité incluses us-west-2a et us-west-2b, vérifiez cela et us-west-2c apparaissent dans la liste.
 6. Si vous ne souhaitez pas accepter ou refuser manuellement chaque demande de connexion envoyée au service de point de terminaison, sous Paramètres supplémentaires, décochez Acceptation requise. Si vous désactivez cette case à cocher, le service de point de terminaison accepte automatiquement chaque demande de connexion qu'il reçoit.
 7. Choisissez Créer.
 8. Dans le nouveau service de point de terminaison, choisissez Autoriser les principaux.
 9. [Contactez-nous](#) pour obtenir l'ARN du point de terminaison VPC Device Farm (ARN du service) à ajouter à la liste des autorisations du service du point de terminaison, puis ajoutez cet ARN de service à la liste des autorisations du service.
 10. Dans l'onglet Détails du service de point de terminaison, notez le nom du service (nom du service). Vous aurez besoin de ce nom lors de la création d'une configuration de point de terminaison de VPC à l'étape suivante.

Votre service de point de terminaison VPC est désormais prêt à être utilisé avec Device Farm.

Étape 3 : Création d'une configuration de point de terminaison VPC dans Device Farm

Après avoir créé un service de point de terminaison dans Amazon VPC, vous pouvez créer une configuration de point de terminaison Amazon VPC dans Device Farm.

1. Connectez-vous à la console Device Farm à l'adresse <https://console.aws.amazon.com/devicefarm>.
2. Dans le volet de navigation, choisissez Mobile device testing, puis Private devices.
3. Choisissez les configurations VPCE.
4. Choisissez Créer une configuration VPCE.

5. Sous Créer une nouvelle configuration VPCE, entrez un nom pour la configuration du point de terminaison VPC.
6. Pour le nom du service VPCE, entrez le nom du service de point de terminaison Amazon VPC (nom du service) que vous avez noté dans la console Amazon VPC. Le nom se présente sous la forme `com.amazonaws.vpce.us-west-2.vpce-svc-id`.
7. Dans le champ Nom DNS du service, entrez le nom DNS du service pour l'application que vous souhaitez tester (par exemple, `devicefarm.com`). Ne spécifiez pas `http` ou `https` avant le nom DNS du service.

Le nom de domaine n'est pas accessible par le biais de l'Internet public. En outre, ce nouveau nom de domaine, qui correspond à votre service de point de terminaison VPC, est généré par Amazon Route 53 et est disponible exclusivement pour vous dans votre session Device Farm.

8. Choisissez Enregistrer.

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

Étape 4 : Création d'un essai

Après avoir enregistré la configuration du point de terminaison VPC, vous pouvez l'utiliser pour créer des tests ou des sessions d'accès à distance. Pour plus d'informations, consultez [Création d'un test dans Device Farm](#) ou [Création d'une session](#).

Journalisation des appels d'API AWS Device Farm avec AWS CloudTrail

AWS Device Farm est intégré à AWS CloudTrail un service qui fournit un enregistrement des actions effectuées par un utilisateur, un rôle ou un AWS service dans AWS Device Farm. CloudTrail capture tous les appels d'API pour AWS Device Farm sous forme d'événements. Les appels capturés incluent des appels provenant de la console AWS Device Farm et des appels de code vers les opérations de l'API AWS Device Farm. Si vous créez un suivi, vous pouvez activer la diffusion continue d'CloudTrail événements vers un compartiment Amazon S3, y compris des événements pour AWS Device Farm. Si vous ne configurez pas de suivi, vous pouvez toujours consulter les événements les plus récents dans la CloudTrail console dans Historique des événements. À l'aide des informations collectées par CloudTrail, vous pouvez déterminer la demande qui a été envoyée à AWS Device Farm, l'adresse IP à partir de laquelle la demande a été faite, l'auteur de la demande, la date à laquelle elle a été faite, ainsi que des informations supplémentaires.

Pour en savoir plus CloudTrail, consultez le [guide de AWS CloudTrail l'utilisateur](#).

Informations sur AWS Device Farm dans CloudTrail

CloudTrail est activé sur votre AWS compte lorsque vous le créez. Lorsqu'une activité a lieu dans AWS Device Farm, cette activité est enregistrée dans un CloudTrail événement avec d'autres événements de AWS service dans l'historique des événements. Vous pouvez consulter, rechercher et télécharger les événements récents dans votre AWS compte. Pour plus d'informations, consultez la section [Affichage des événements à l'aide de l'historique des CloudTrail événements](#).

Pour un enregistrement continu des événements de votre AWS compte, y compris des événements pour AWS Device Farm, créez un historique. Un suivi permet CloudTrail de fournir des fichiers journaux à un compartiment Amazon S3. Par défaut, lorsque vous créez un journal d'activité dans la console, il s'applique à toutes les régions AWS. Le journal enregistre les événements de toutes les régions de la AWS partition et transmet les fichiers journaux au compartiment Amazon S3 que vous spécifiez. En outre, vous pouvez configurer d'autres AWS services pour analyser plus en détail les données d'événements collectées dans les CloudTrail journaux et agir en conséquence. Pour plus d'informations, consultez les ressources suivantes :

- [Vue d'ensemble de la création d'un journal d'activité](#)
- [CloudTrail Services et intégrations pris en charge](#)

- [Configuration des notifications Amazon SNS pour CloudTrail](#)
- [Réception de fichiers CloudTrail journaux de plusieurs régions](#) et [réception de fichiers CloudTrail journaux de plusieurs comptes](#)

Lorsque la CloudTrail connexion est activée dans votre AWS compte, les appels d'API effectués aux actions de Device Farm sont suivis dans des fichiers journaux. Les enregistrements Device Farm sont écrits avec les autres enregistrements de AWS service dans un fichier journal. CloudTrail détermine à quel moment créer et écrire dans un nouveau fichier en fonction d'une période et de la taille du fichier.

Toutes les actions de Device Farm sont enregistrées et documentées dans le [AWS CLI référence](#) et le [Automatiser Device Farm](#). Par exemple, les appels à créer un nouveau projet ou à exécuter dans Device Farm génèrent des entrées dans des fichiers CloudTrail journaux.

Chaque événement ou entrée de journal contient des informations sur la personne ayant initié la demande. Les informations relatives à l'identité permettent de déterminer les éléments suivants :

- Si la demande a été faite avec les informations d'identification de l'utilisateur root ou Gestion des identités et des accès AWS (IAM).
- Si la demande a été effectuée avec les informations d'identification de sécurité temporaires d'un rôle ou d'un utilisateur fédéré.
- Si la demande a été faite par un autre AWS service.

Pour plus d'informations, consultez la section [Élément userIdentity CloudTrail](#).

Comprendre les entrées du fichier journal AWS Device Farm

Un suivi est une configuration qui permet de transmettre des événements sous forme de fichiers journaux à un compartiment Amazon S3 que vous spécifiez. CloudTrail les fichiers journaux contiennent une ou plusieurs entrées de journal. Un événement représente une demande unique provenant de n'importe quelle source et inclut des informations sur l'action demandée, la date et l'heure de l'action, les paramètres de la demande, etc. CloudTrail les fichiers journaux ne constituent pas une trace ordonnée des appels d'API publics, ils n'apparaissent donc pas dans un ordre spécifique.

L'exemple suivant montre une entrée de CloudTrail journal illustrant l'`ListRuns` action Device Farm :

```

{
  "Records": [
    {
      "eventVersion": "1.03",
      "userIdentity": {
        "type": "Root",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:root",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2015-07-08T21:13:35Z"
          }
        }
      },
      "eventTime": "2015-07-09T00:51:22Z",
      "eventSource": "devicefarm.amazonaws.com",
      "eventName": "ListRuns",
      "awsRegion": "us-west-2",
      "sourceIPAddress": "203.0.113.11",
      "userAgent": "example-user-agent-string",
      "requestParameters": {
        "arn": "arn:aws:devicefarm:us-west-2:123456789012:project:a9129b8c-df6b-4cdd-8009-40a25EXAMPLE"},
      "responseElements": {
        "runs": [
          {
            "created": "Jul 8, 2015 11:26:12 PM",
            "name": "example.apk",
            "completedJobs": 2,
            "arn": "arn:aws:devicefarm:us-west-2:123456789012:run:a9129b8c-df6b-4cdd-8009-40a256aEXAMPLE/1452d105-e354-4e53-99d8-6c993EXAMPLE",
            "counters": {
              "stopped": 0,
              "warned": 0,
              "failed": 0,
              "passed": 4,
              "skipped": 0,
              "total": 4,
              "errored": 0
            }
          },

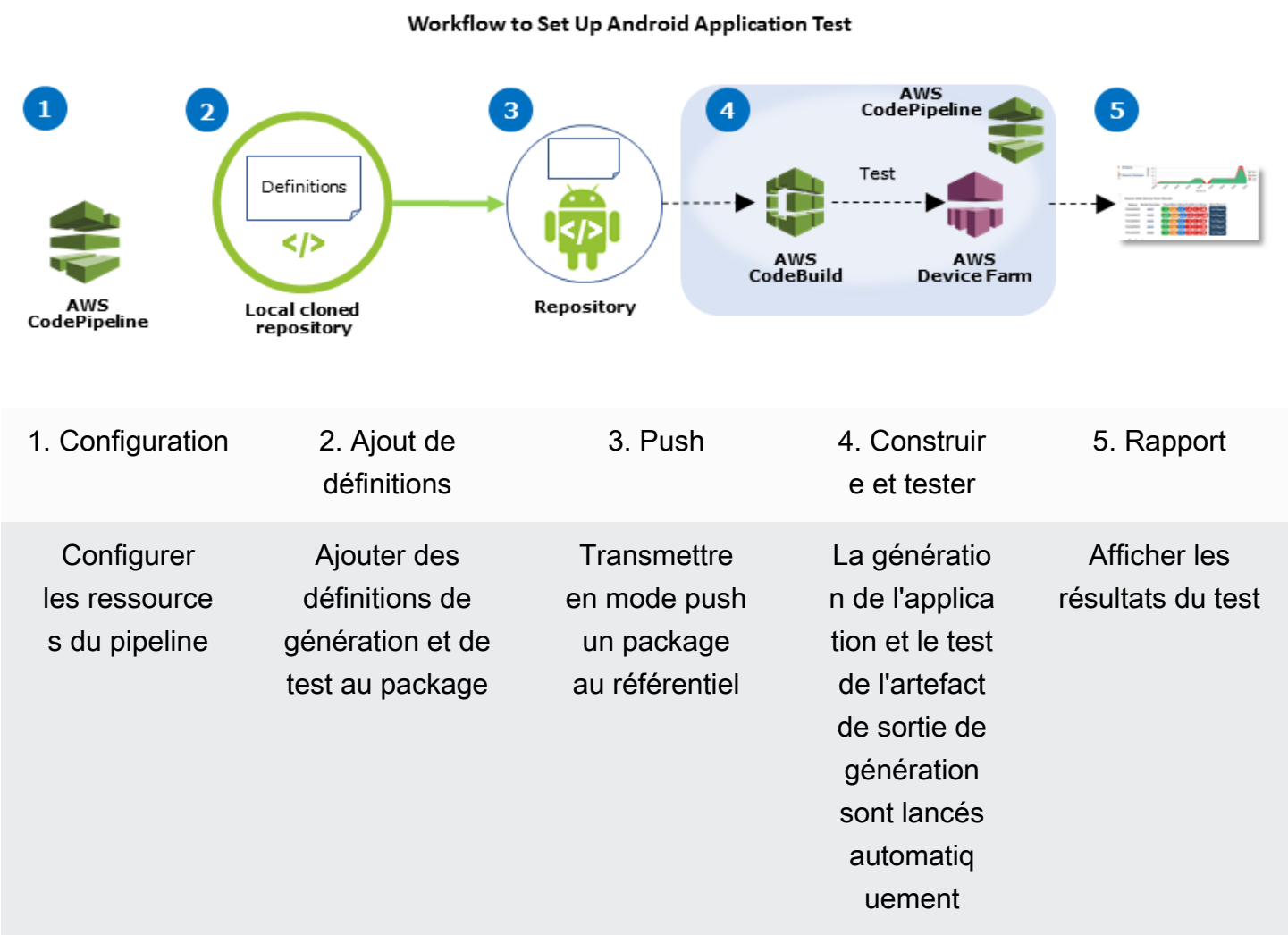
```

```
        "type": "BUILTIN_FUZZ",
        "status": "RUNNING",
        "totalJobs": 3,
        "platform": "ANDROID_APP",
        "result": "PENDING"
    },
    ... additional entries ...
]
}
}
}
]
```

Intégration d'AWS Device Farm dans une phase CodePipeline de test

Vous pouvez l'utiliser [AWS CodePipeline](#) pour intégrer les tests d'applications mobiles configurés dans Device Farm dans un pipeline de publication automatisé géré par AWS. Vous pouvez configurer votre pipeline de manière à ce qu'il exécute des tests à la demande, en fonction d'un calendrier ou dans le cadre d'une intégration continue.

Le schéma suivant illustre le flux d'intégration continue dans lequel une application Android est conçue et testée chaque fois qu'une notification push est validée pour son référentiel. Pour créer cette configuration de pipeline, consultez le [didacticiel : Création et test d'une application Android lorsqu'elle est poussée vers GitHub](#).



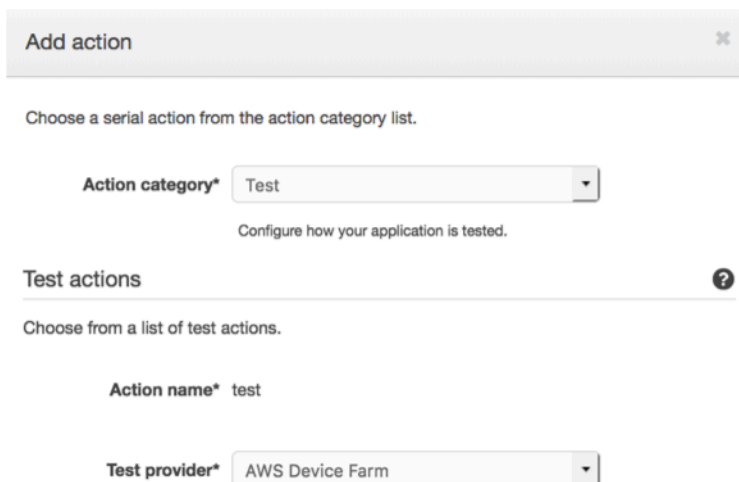
Pour savoir comment configurer un pipeline qui teste en continu une application compilée (par exemple, un fichier iOS .ipa ou un fichier Android .apk) en tant que source, consultez [Didacticiel : Test d'une application iOS à chaque chargement d'un fichier .ipa vers un compartiment Amazon S3](#).

Configurez CodePipeline pour utiliser vos tests Device Farm

Dans ces étapes, nous partons du principe que vous avez [configuré un projet Device Farm](#) et [créé un pipeline](#). Le pipeline doit être configuré avec une étape de test recevant un [artefact d'entrée](#) qui contient votre définition de test et les fichiers du package de l'application compilée. L'artefact d'entrée de l'étape de test peut être l'artefact de sortie d'une source ou d'une étape de génération configurée dans votre pipeline.

Pour configurer un test Device Farm, exécutez-le en tant qu'action de CodePipeline test

1. Connectez-vous à la CodePipeline console AWS Management Console et ouvrez-la à l'adresse <https://console.aws.amazon.com/codepipeline/>.
2. Choisissez le pipeline de la version de votre application.
3. Dans le panneau de l'étape de test, choisissez l'icône en forme de crayon, puis Action.
4. Dans le panneau Ajouter une action, pour Catégorie d'action, choisissez Test.
5. Dans Nom de l'action, entrez un nom.
6. Dans Fournisseur du test, choisissez AWS Device Farm.



The screenshot shows the 'Add action' dialog in the AWS CodePipeline console. At the top, there's a header 'Add action' with a close button. Below it, a message says 'Choose a serial action from the action category list.' The 'Action category*' dropdown is set to 'Test'. Below this, a message says 'Configure how your application is tested.' The 'Test actions' section has a header and a help icon. Below it, a message says 'Choose from a list of test actions.' The 'Action name*' field contains the text 'test'. The 'Test provider*' dropdown is set to 'AWS Device Farm'.

7. Dans Nom du projet, choisissez votre projet Device Farm existant ou choisissez Create a new project.

8. Dans Groupe d'appareils, choisissez votre groupe d'appareils existant, ou bien Créer un nouveau groupe d'appareils. Si vous créez un groupe d'appareils, vous devez sélectionner un ensemble d'appareils de test.
9. Dans Type d'application, choisissez la plateforme correspondant à votre application.

Device Farm Test

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

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

10. Dans Chemin du fichier d'application, entrez le chemin d'accès au package de l'application compilée. Ce chemin dépend de la racine de l'artefact d'entrée de votre test.
11. Dans Type de test, effectuez l'une des actions suivantes :
 - Si vous utilisez l'un des tests Device Farm intégrés, choisissez le type de test configuré dans votre projet Device Farm.
 - Si vous n'utilisez pas l'un des tests intégrés à Device Farm, dans le chemin du fichier de test, entrez le chemin du fichier de définition du test. Ce chemin dépend de la racine de l'artefact d'entrée de votre test.

The image shows three overlapping screenshots of the AWS Device Farm configuration interface, illustrating different test types and their associated settings.

- Top screenshot (Calabash):** Shows the 'Test type*' dropdown set to 'Calabash'. The 'Test file path' is 'tests.zip'.
- Middle screenshot (Appium Java TestNG):** Shows the 'Test type*' dropdown set to 'Appium Java TestNG'. The 'Test file path' is 'tests.zip' and the 'Appium version' is '1.7.2'.
- Bottom screenshot (Built-in: Fuzz):** Shows the 'Test type*' dropdown set to 'Built-in: Fuzz'. The 'Event count' is '6000' and the 'Event throttle' is '50'. The 'Randomizer seed' field is empty.

12. Dans les autres champs, indiquez la configuration appropriée pour votre test et le type d'application.
13. (Facultatif) Dans Avancé, indiquez la configuration détaillée de votre test.

▼ Advanced

Device artifacts

Location on the device where custom artifacts will be stored.

Host machine artifacts

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

Add extra data

Location of extra data needed for this test.

Execution timeout

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

Latitude

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

Longitude

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

Set Radio Stats

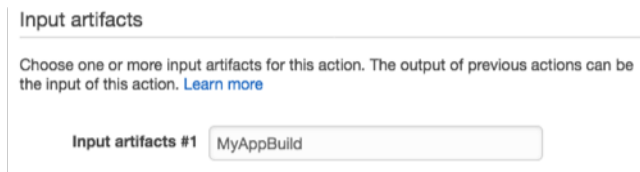
Bluetooth ☒ GPS ☒

NFC ☒ Wifi ☒

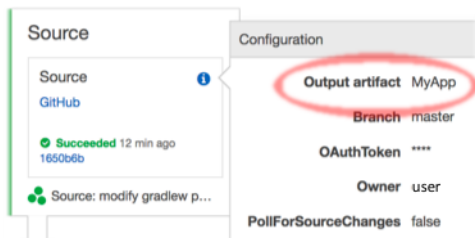
Enable app performance data capture ☒ Enable video recording ☒

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

14. Dans Artefacts d'entrée, choisissez l'artefact d'entrée correspondant à l'artefact de sortie de l'étape qui précède l'étape de test dans le pipeline.



Dans la CodePipeline console, vous pouvez trouver le nom de l'artefact de sortie pour chaque étape en survolant l'icône d'information dans le diagramme du pipeline. Si votre pipeline teste votre application directement depuis l'étape Source, choisissez MyApp. Si votre pipeline inclut une phase de construction, choisissez MyAppBuild.



15. Au bas du panneau, choisissez Ajouter une action.
16. Dans le CodePipeline volet, choisissez Enregistrer la modification du pipeline, puis sélectionnez Enregistrer la modification.
17. Pour soumettre vos modifications et lancer la génération d'un pipeline, choisissez Changement de version, puis Publication.

AWS CLI référence pour AWS Device Farm

Pour utiliser le AWS Command Line Interface (AWS CLI) pour exécuter des commandes Device Farm, consultez la [AWS CLI référence relative à AWS Device Farm](#).

Pour obtenir des informations générales sur le AWS CLI, consultez le [guide de l'AWS Command Line Interface utilisateur](#) et le manuel de [référence des AWS CLI commandes](#).

PowerShell Référence Windows pour AWS Device Farm

[Pour utiliser Windows PowerShell pour exécuter des commandes Device Farm, consultez le manuel Device Farm Cmdlet Reference dans le AWS Tools for Windows PowerShell manuel Cmdlet Reference.](#) Pour plus d'informations, consultez la section [Configuration des outils AWS pour Windows PowerShell](#) dans le guide de Outils AWS pour PowerShell l'utilisateur.

Automatisation d'AWS Device Farm

L'accès programmatique à Device Farm est un moyen puissant d'automatiser les tâches courantes que vous devez accomplir, telles que la planification d'une exécution ou le téléchargement des artefacts pour une exécution, une suite ou un test. Le AWS SDK et le fournisseur AWS CLI permettent de le faire.

Le AWS SDK permet d'accéder à tous les AWS services, notamment Device Farm, Amazon S3, etc. Pour plus d'informations, veuillez consulter la rubrique

- les [AWS outils et SDKs](#)
- la [référence de l'API AWS Device Farm](#)

Exemple : utilisation de la AWS CLI ou du SDK pour télécharger une application ou effectuer un test sur Device Farm

Les exemples suivants montrent comment créer un téléchargement sur Device Farm à l'aide de la AWS CLI ou du AWS SDK dans différentes langues. Les téléchargements sont les éléments de base de la planification des tests sur Device Farm. Ils incluent les éléments suivants :

- Votre application
- Votre test
- Votre fichier [de spécifications de test](#)

Les téléchargements sont créés à l'aide de l'[CreateUpload](#) API. Cette API renvoie une URL présignée S3 vers laquelle vous pouvez transférer votre téléchargement à l'aide d'une requête HTTP PUT. L'URL expire au bout de 24 heures.

AWS CLI

Remarque : cet exemple utilise l'[outil de ligne de commande](#) pour transférer l'application *curl* vers Device Farm.

Créez d'abord un projet si ce n'est pas déjà fait.

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

Cela affichera des résultats tels que les suivants :

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

Procédez ensuite comme suit pour créer votre téléchargement et le transférer vers Device Farm. Dans cet exemple, nous allons créer un téléchargement d'application Android à l'aide d'un fichier APK local. Pour plus d'informations sur les types de téléchargement, y compris des détails sur les types de téléchargement d'applications iOS, consultez notre documentation d'API pour créer un [Upload](#).

```
$ export APP_PATH="/local/path/to/my_sample_app.apk"
$ export APP_TYPE="ANDROID_APP"
```

Tout d'abord, nous créons le téléchargement dans Device Farm :

```
$ aws devicefarm create-upload \
  --project-arn "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE" \
  --name "${basename "$APP_PATH"}" \
  --type "$APP_TYPE"
```

Cela affichera des résultats tels que les suivants :

```
{
  "upload": {
    "arn": "arn:aws:devicefarm:us-
west-2:385076942068:upload:490a6350-0ba3-43e5-83f5-d2896b069a34/a120e848-c57b-4e8d-
a720-d750a0c4d936",
    "name": "my_sample_app.apk",
    "created": 1760747318.266,
    "type": "ANDROID_APP",
    "status": "INITIALIZED",
    "url": "https://prod-us-west-2-uploads.s3.dualstack.us-west-2.amazonaws.com/
arn%3Aaws%3Adevicefarm%3Aus-west-2..."
```

```

    "category": "PRIVATE"
  }
}

```

Effectuez ensuite un appel PUT à l'aide de curl pour transférer l'application vers le compartiment S3 de Device Farm :

```
$ curl -T "$APP_PATH" "https://prod-us-west-2-uploads.s3.dualstack.us-west-2.amazonaws.com/arn%3Aaws%3Adevicefarm%3Aus-west-2..."
```

Enfin, attendez que le statut de l'application soit « réussi » :

```
$ aws devicefarm get-upload --arn "arn:aws:devicefarm:us-west-2:385076942068:upload:490a6350-0ba3-43e5-83f5-d2896b069a34/a120e848-c57b-4e8d-a720-d750a0c4d936"
```

Cela affichera des résultats tels que les suivants :

```

{
  "upload": {
    "arn": "arn:aws:devicefarm:us-west-2:385076942068:upload:490a6350-0ba3-43e5-83f5-d2896b069a34/a120e848-c57b-4e8d-a720-d750a0c4d936",
    "name": "my_sample_app.apk",
    "created": 1760747318.266,
    "type": "ANDROID_APP",
    "status": "SUCCEEDED",
    "url": "https://prod-us-west-2-uploads.s3.dualstack.us-west-2.amazonaws.com/arn%3Aaws%3Adevicefarm%3Aus-west-2...",
    "metadata": "{\"activity_name\": \"com.amazonaws.devicefarm.android.referenceapp.Activities.MainActivity\", \"package_name\": \"com.amazonaws.devicefarm.android.referenceapp\", ...}",
    "category": "PRIVATE"
  }
}

```

Python

Remarque : cet exemple utilise le *requests* package tiers pour transférer l'application vers Device Farm, ainsi que le AWS SDK pour Python *boto3*.

Créez d'abord un projet si ce n'est pas déjà fait.

```
import boto3

client = boto3.client("devicefarm", region_name="us-west-2")
resp = client.create_project(name="MyProjectName")

print(resp)
# Response will be something like:
# {
#     "project": {
#         "name": "MyProjectName",
#         "arn": "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE",
#         "created": 1535675814.414
#     }
# }
```

Procédez ensuite comme suit pour créer votre téléchargement et le transférer vers Device Farm. Dans cet exemple, nous allons créer un téléchargement d'application Android à l'aide d'un fichier APK local. Pour plus d'informations sur les types de téléchargement, y compris des détails sur les types de téléchargement d'applications iOS, consultez notre documentation d'API pour créer un [Upload](#).

```
import os
import time
import datetime
import requests
from pathlib import Path
import boto3

def upload_device_farm_file():
    project_arn = "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE"
    app_path = Path("/local/path/to/my_sample_app.apk")
    file_type = "ANDROID_APP"

    if not app_path.is_file():
        raise RuntimeError(f"{app_path} is not a valid app file path")

    client = boto3.client("devicefarm", region_name="us-west-2")

    # 1) Create the upload in Device Farm
```

```

create = client.create_upload(
    projectArn=project_arn,
    name=app_path.name,
    type=file_type,
    contentType="application/octet-stream",
)
upload = create["upload"]
upload_arn = upload["arn"]
upload_url = upload["url"]
# This will show output such as the following:
# { "upload": { "arn": "...", "name": "my_sample_app.apk", "type":
"ANDROID_APP", "status": "INITIALIZED", "url": "https://..." } }

# 2) Do an HTTP PUT command to push the file to the pre-signed S3 URL
with app_path.open("rb") as fh:
    print(f"Uploading {app_path.name} to Device Farm...")
    put_resp = requests.put(upload_url, data=fh, headers={"Content-Type":
"application/octet-stream"})
    put_resp.raise_for_status()

# 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
timeout_seconds = 30
start = time.time()
while True:
    get_resp = client.get_upload(arn=upload_arn)
    status = get_resp["upload"]["status"]
    msg = get_resp["upload"].get("message") or
get_resp["upload"].get("metadata") or ""
    elapsed = datetime.timedelta(seconds=int(time.time() - start))
    print(f"[{elapsed}] status={status}{' - ' + msg if msg else ''}")

    if status == "SUCCEEDED":
        print(f"Upload complete: {upload_arn}")
        return upload_arn
    if status == "FAILED":
        raise RuntimeError(f"Device Farm failed to process upload: {msg}")

    if (time.time() - start) > timeout_seconds:
        raise RuntimeError(f"Timed out after {timeout_seconds}s waiting for
upload to process (last status={status}).")

    time.sleep(1)

```

```
upload_device_farm_file()
```

Java

Remarque : cet exemple utilise le AWS SDK pour Java v2 *HttpClient* et pour transférer l'application vers Device Farm. Il est compatible avec les versions 11 et supérieures du JDK.

Créez d'abord un projet si ce n'est pas déjà fait.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import software.amazon.awssdk.services.devicefarm.model.CreateProjectRequest;
import software.amazon.awssdk.services.devicefarm.model.CreateProjectResponse;

try (DeviceFarmClient client = DeviceFarmClient.builder()
    .region(Region.US_WEST_2)
    .build()) {
    CreateProjectResponse resp = client.createProject(
        CreateProjectRequest.builder().name("MyProjectName").build());
    System.out.println(resp.project());
    // Response will be something like:
    // Project{name=MyProjectName, arn=arn:aws:devicefarm:us-
    west-2:123456789101:project:5e01a8c7-..., created=...}
}
```

Procédez ensuite comme suit pour créer votre téléchargement et le transférer vers Device Farm. Dans cet exemple, nous allons créer un téléchargement d'application Android à l'aide d'un fichier APK local. Pour plus d'informations sur les types de téléchargement, y compris des détails sur les types de téléchargement d'applications iOS, consultez notre documentation d'API pour créer un [Upload](#).

```
import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.time.Duration;
import java.time.Instant;
```



```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import software.amazon.awssdk.services.devicefarm.model.CreateUploadRequest;
import software.amazon.awssdk.services.devicefarm.model.CreateUploadResponse;
import software.amazon.awssdk.services.devicefarm.model.GetUploadRequest;
import software.amazon.awssdk.services.devicefarm.model.GetUploadResponse;
import software.amazon.awssdk.services.devicefarm.model.Upload;
import software.amazon.awssdk.services.devicefarm.model.UploadType;

public class DeviceFarmUploader {

    public static String upload(String projectArn, Path appPath) throws Exception {
        if (projectArn == null || projectArn.isEmpty()) {
            throw new IllegalArgumentException("Missing projectArn");
        }
        if (!Files.isRegularFile(appPath)) {
            throw new IllegalArgumentException("Invalid app path: " + appPath);
        }

        String fileName = appPath.getFileName().toString().trim();
        UploadType type = UploadType.ANDROID_APP;

        // Build a reusable HttpClient
        HttpClient http = HttpClient.newBuilder()
            .version(HttpClient.Version.HTTP_1_1)
            .connectTimeout(Duration.ofSeconds(10))
            .build();

        try (DeviceFarmClient client = DeviceFarmClient.builder()
            .region(Region.US_WEST_2)
            .build()) {

            // 1) Create the upload in Device Farm
            CreateUploadResponse create =
client.createUpload(CreateUploadRequest.builder()
                .projectArn(projectArn)
                .name(fileName)
                .type(type)
                .contentType("application/octet-stream")
                .build());

            Upload upload = create.upload();
            String uploadArn = upload.arn();
            String url = upload.url();
```

```

        // This will show output such as the following:
        // { "upload": { "arn": "...", "name": "my_sample_app.apk", "type":
"ANDROID_APP", "status": "INITIALIZED", "url": "https://..." } }

        // 2) PUT file to pre-signed URL using HttpClient
        HttpRequest put = HttpRequest.newBuilder(URI.create(url))
            .timeout(Duration.ofMinutes(15))
            .header("Content-Type", "application/octet-stream")
            .PUT(HttpRequest.BodyPublishers.ofFile(appPath))
            .build();

        HttpResponse<Void> resp = http.send(put,
        HttpResponse.BodyHandlers.discarding());
        int code = resp.statusCode();
        if (code / 100 != 2) {
            throw new IOException("Failed PUT to S3 pre-signed URL, HTTP " +
code);
        }

        // 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
        Instant deadline = Instant.now().plusSeconds(30); // 30-second timeout
        while (true) {
            GetUploadResponse got = client.getUpload(GetUploadRequest.builder()
                .arn(uploadArn)
                .build());

            String status = got.upload().statusAsString();
            String msg = got.upload().metadata();
            System.out.println("status=" + status + (msg != null ? " - " + msg :
""));

            if ("SUCCEEDED".equals(status)) return uploadArn;
            if ("FAILED".equals(status)) throw new RuntimeException("Upload
failed: " + msg);
            if (Instant.now().isAfter(deadline)) {
                throw new RuntimeException("Timeout waiting for processing, last
status=" + status);
            }
            Thread.sleep(2000);
        }
    }
}

public static void main(String[] args) throws Exception {

```

```
String projectArn = "arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-c861-4c0a-b1d5-12345EXAMPLE";
Path appPath = Paths.get("/local/path/to/my_sample_app.apk");
String result = upload(projectArn, appPath);
System.out.println("Upload ARN: " + result);
}
}
```

JavaScript

Remarque : cet exemple utilise le AWS SDK pour JavaScript (v3) et Node 18+ *fetch* pour transférer l'application vers Device Farm.

Créez d'abord un projet si ce n'est pas déjà fait.

```
import { DeviceFarmClient, CreateProjectCommand } from "@aws-sdk/client-device-
farm";

const df = new DeviceFarmClient({ region: "us-west-2" });
const resp = await df.send(new CreateProjectCommand({ name: "MyProjectName" }));
console.log(resp);
// Response will be something like:
// { project: { name: 'MyProjectName', arn: 'arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-...', created: 1535675814.414 } }
```

Procédez ensuite comme suit pour créer votre téléchargement et le transférer vers Device Farm. Dans cet exemple, nous allons créer un téléchargement d'application Android à l'aide d'un fichier APK local. Pour plus d'informations sur les types de téléchargement, y compris des détails sur les types de téléchargement d'applications iOS, consultez notre documentation d'API pour créer un [Upload](#).

```
import { DeviceFarmClient, CreateUploadCommand, GetUploadCommand } from "@aws-sdk/
client-device-farm";
import { createReadStream } from "fs";
import { basename } from "path";

const projectArn = "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE";
const appPath = "/local/path/to/my_sample_app.apk";
const name = basename(appPath).trim();
const type = "ANDROID_APP";
```

```

const client = new DeviceFarmClient({ region: "us-west-2" });

// 1) Create the upload in Device Farm
const create = await client.send(new CreateUploadCommand({
  projectArn,
  name,
  type,
  contentType: "application/octet-stream",
}));

const uploadArn = create.upload.arn;
const url = create.upload.url;
// This will show output such as the following:
// { upload: { arn: '...', name: 'my_sample_app.apk', type: 'ANDROID_APP', status:
// 'INITIALIZED', url: 'https://...' } }

// 2) PUT to pre-signed URL
const putResp = await fetch(url, {
  method: "PUT",
  headers: { "Content-Type": "application/octet-stream" },
  body: createReadStream(appPath),
});
if (!putResp.ok) {
  throw new Error(`Failed PUT to pre-signed URL: ${putResp.status} ${await
putResp.text().catch(()=>"")}`);
}

// 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
const deadline = Date.now() + (30 * 1000); // 30-second timeout
while (true) {
  const response = await client.send(new GetUploadCommand({ arn: uploadArn }));
  const { status, message, metadata } = response.upload;
  console.log(`status=${status}${message ? " - " + message : metadata ? " - " +
metadata : ""}`);
  if (status === "SUCCEEDED") {
    console.log("Upload complete:", uploadArn);
    break;
  }
  if (status === "FAILED") {
    throw new Error(`Upload failed: ${message || metadata || "unknown"}`);
  }
  if (Date.now() > deadline) throw new Error(`Timeout waiting for processing (last
status=${status})`);
  await new Promise(r => setTimeout(r, 2000));
}

```

```
}
```

C#

Remarque : cet exemple utilise le AWS SDK pour .NET *HttpClient* et permet de transférer l'application vers Device Farm.

Créez d'abord un projet si ce n'est pas déjà fait.

```
using System;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

using var client = new AmazonDeviceFarmClient(RegionEndpoint.USWest2);
var resp = await client.CreateProjectAsync(new CreateProjectRequest { Name =
    "MyProjectName" });
Console.WriteLine(resp.Project);
// Response will be something like:
// { Name = MyProjectName, Arn = arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-..., Created = ... }
```

Procédez ensuite comme suit pour créer votre téléchargement et le transférer vers Device Farm. Dans cet exemple, nous allons créer un téléchargement d'application Android à l'aide d'un fichier APK local. Pour plus d'informations sur les types de téléchargement, y compris des détails sur les types de téléchargement d'applications iOS, consultez notre documentation d'API pour créer un [Upload](#).

```
using System;
using System.IO;
using System.Net.Http;
using System.Threading.Tasks;
using System.Net.Http.Headers;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

class DeviceFarmUploader
{
    public static async Task<string> UploadAsync(string projectArn, string appPath)
    {
```

```

        if (string.IsNullOrEmpty(projectArn)) throw new
ArgumentException("Missing projectArn");
        if (!File.Exists(appPath)) throw new ArgumentException($"Invalid app path:
{appPath}");
        var type = UploadType.ANDROID_APP;

        using var client = new AmazonDeviceFarmClient(RegionEndpoint.USWest2);
        // 1) Create the upload in Device Farm
        var create = await client.CreateUploadAsync(new CreateUploadRequest
        {
            ProjectArn = projectArn,
            Name = Path.GetFileName(appPath),
            Type = type,
            ContentType = "application/octet-stream"
        });

        var uploadArn = create.Upload.Arn;
        var url = create.Upload.Url;
        // This will show output such as the following:
        // { Upload: { Arn = ..., Name = my_sample_app.apk, Type = ANDROID_APP,
Status = INITIALIZED, Url = https://... } }

        // 2) PUT file to pre-signed URL
        using (var http = new HttpClient())
        using (var fs = File.OpenRead(appPath))
        using (var content = new StreamContent(fs))
        {
            content.Headers.Add("Content-Type", "application/octet-stream");
            var resp = await http.PutAsync(url, content);
            if (!resp.IsSuccessStatusCode)
                throw new Exception($"Failed PUT to pre-signed URL:
{(int)resp.StatusCode} {await resp.Content.ReadAsStringAsync()}");
        }

        // 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
        var deadline = DateTime.UtcNow.AddSeconds(30); // 30-second timeout
        while (true)
        {
            var got = await client.GetUploadAsync(new GetUploadRequest { Arn =
uploadArn });
            var status = got.Upload.Status.Value;
            var msg = got.Upload.Message ?? got.Upload.Metadata;
            Console.WriteLine($"status={status}{(string.IsNullOrEmpty(msg) ? "" : "
- " + msg)}");

```

```

        if (status == UploadStatus.SUCCEEDED.Value) return uploadArn;
        if (status == UploadStatus.FAILED.Value) throw new Exception($"Upload
failed: {msg}");
        if (DateTime.UtcNow > deadline) throw new TimeoutException($"Timeout
waiting for processing (last status={status})");
        await Task.Delay(2000);
    }
}

static async Task Main()
{
    var projectArn = "arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-c861-4c0a-b1d5-12345EXAMPLE";
    var appPath = "/local/path/to/my_sample_app.apk";
    var result = await UploadAsync(projectArn!, appPath!);
    Console.WriteLine("Upload ARN: " + result);
}
}

```

Ruby

Remarque : cet exemple utilise le AWS SDK pour Ruby *Net::HTTP* et permet de transférer l'application vers Device Farm.

Créez d'abord un projet si ce n'est pas déjà fait.

```

require "aws-sdk-devicefarm"

client = Aws::DeviceFarm::Client.new(region: "us-west-2")
resp = client.create_project(name: "MyProjectName")
puts resp.project.inspect
# Response will be something like:
# #<struct Aws::DeviceFarm::Types::Project name="MyProjectName",
#   arn="arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-...",
#   created=1535675814.414>

```

Procédez ensuite comme suit pour créer votre téléchargement et le transférer vers Device Farm. Dans cet exemple, nous allons créer un téléchargement d'application Android à l'aide d'un fichier APK local. Pour plus d'informations sur les types de téléchargement, y compris des détails sur les types de téléchargement d'applications iOS, consultez notre documentation d'API pour créer un [Upload](#).

```

require "aws-sdk-devicefarm"
require "net/http"
require "uri"

project_arn = "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-c861-4c0a-b1d5-12345EXAMPLE"
app_path    = "/local/path/to/my_sample_app.apk"
raise "Invalid APP_PATH: #{app_path}" unless File.file?(app_path)
type = "ANDROID_APP"

client = Aws::DeviceFarm::Client.new(region: "us-west-2")

# 1) Create the upload in Device Farm
create = client.create_upload(
  project_arn: project_arn,
  name: File.basename(app_path),
  type: type,
  content_type: "application/octet-stream"
)

upload_arn = create.upload.arn
url = create.upload.url
# This will show output such as the following:
# #<Upload arn="...", name="my_sample_app.apk", type="ANDROID_APP",
#   status="INITIALIZED", url="https://...">

# 2) PUT the file to the pre-signed URL
uri = URI.parse(url)
Net::HTTP.start(uri.host, uri.port, use_ssl: (uri.scheme == "https")) do |http|
  req = Net::HTTP::Put.new(uri)
  req["Content-Type"] = "application/octet-stream"
  req.body_stream = File.open(app_path, "rb")
  req.content_length = File.size(app_path)
  resp = http.request(req)
  raise "Failed PUT: #{resp.code} #{resp.body}" unless resp.code.to_i / 100 == 2
end

# 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
deadline = Time.now + 30 # 30-second timeout
loop do
  got = client.get_upload(arn: upload_arn)
  status = got.upload.status
  msg = got.upload.message || got.upload.metadata

```



```
puts "status=#{status}#{msg ? " - #{msg}" : ""}"

case status
when "SUCCEEDED" then puts "Upload complete: #{upload_arn}"; break
when "FAILED"     then raise "Upload failed: #{msg}"
end
raise "Timeout waiting for processing (last status=#{status})" if Time.now >
deadline
sleep 2
end
```

Exemple : utilisation du AWS SDK pour démarrer l'exécution d'une Device Farm et collecter des artefacts

L'exemple suivant montre beginning-to-end comment utiliser le AWS SDK pour travailler avec Device Farm. Cet exemple effectue les opérations suivantes :

- Télécharge un package de test et d'application sur Device Farm
- Démarre un test et attend qu'il se termine (ou qu'il échoue)
- Télécharge tous les artefacts produits par les suites de test

Cet exemple dépend du package `requests` tiers pour interagir avec HTTP.

```
import boto3
import os
import requests
import string
import random
import time
import datetime
import time
import json

# The following script runs a test through Device Farm
#
# Things you have to change:
config = {
    # This is our app under test.
    "appFilePath": "app-debug.apk",
```

```

    "projectArn": "arn:aws:devicefarm:us-
west-2:111122223333:project:1b99bcff-1111-2222-ab2f-8c3c733c55ed",
    # Since we care about the most popular devices, we'll use a curated pool.
    "testSpecArn": "arn:aws:devicefarm:us-west-2::upload:101e31e8-12ac-11e9-ab14-
d663bd873e83",
    "poolArn": "arn:aws:devicefarm:us-west-2::devicepool:082d10e5-d7d7-48a5-ba5c-
b33d66efa1f5",
    "namePrefix": "MyAppTest",
    # This is our test package. This tutorial won't go into how to make these.
    "testPackage": "tests.zip"
}

client = boto3.client('devicefarm')

unique =
    config['namePrefix']+"-"+(datetime.date.today().isoformat())+(''.join(random.sample(string.ascii_letters, 10)))

print(f"The unique identifier for this run is going to be {unique} -- all uploads will
be prefixed with this.")

def upload_df_file(filename, type_, mime='application/octet-stream'):
    response = client.create_upload(projectArn=config['projectArn'],
        name = (unique)+"_"+os.path.basename(filename),
        type=type_,
        contentType=mime
    )
    # Get the upload ARN, which we'll return later.
    upload_arn = response['upload']['arn']
    # We're going to extract the URL of the upload and use Requests to upload it
    upload_url = response['upload']['url']
    with open(filename, 'rb') as file_stream:
        print(f"Uploading {filename} to Device Farm as {response['upload']['name']}...
",end='')
        put_req = requests.put(upload_url, data=file_stream, headers={"content-
type":mime})
        print(' done')
        if not put_req.ok:
            raise Exception("Couldn't upload, requests said we're not ok. Requests
says: "+put_req.reason)
        started = datetime.datetime.now()
        while True:
            print(f"Upload of {filename} in state {response['upload']['status']} after
"+str(datetime.datetime.now() - started))
            if response['upload']['status'] == 'FAILED':

```

```

        raise Exception("The upload failed processing. DeviceFarm says reason
is: \n" + (response['upload']['message'] if 'message' in response['upload'] else
response['upload']['metadata']))
        if response['upload']['status'] == 'SUCCEEDED':
            break
        time.sleep(5)
        response = client.get_upload(arn=upload_arn)
    print("")
    return upload_arn

our_upload_arn = upload_df_file(config['appFilePath'], "ANDROID_APP")
our_test_package_arn = upload_df_file(config['testPackage'],
    'APPIUM_PYTHON_TEST_PACKAGE')
print(our_upload_arn, our_test_package_arn)
# Now that we have those out of the way, we can start the test run...
response = client.schedule_run(
    projectArn = config["projectArn"],
    appArn = our_upload_arn,
    devicePoolArn = config["poolArn"],
    name=unique,
    test = {
        "type": "APPIUM_PYTHON",
        "testSpecArn": config["testSpecArn"],
        "testPackageArn": our_test_package_arn
    }
)
run_arn = response['run']['arn']
start_time = datetime.datetime.now()
print(f"Run {unique} is scheduled as arn {run_arn} ")

try:

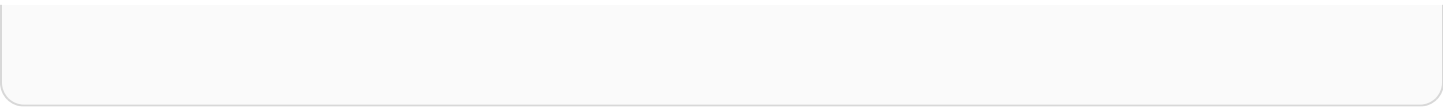
    while True:
        response = client.get_run(arn=run_arn)
        state = response['run']['status']
        if state == 'COMPLETED' or state == 'ERRORED':
            break
        else:
            print(f" Run {unique} in state {state}, total time
"+str(datetime.datetime.now()-start_time))
            time.sleep(10)
except:
    # If something goes wrong in this process, we stop the run and exit.

```

```

    client.stop_run(arn=run_arn)
    exit(1)
print(f"Tests finished in state {state} after "+str(datetime.datetime.now() -
    start_time))
# now, we pull all the logs.
jobs_response = client.list_jobs(arn=run_arn)
# Save the output somewhere. We're using the unique value, but you could use something
    else
save_path = os.path.join(os.getcwd(), unique)
os.mkdir(save_path)
# Save the last run information
for job in jobs_response['jobs'] :
    # Make a directory for our information
    job_name = job['name']
    os.makedirs(os.path.join(save_path, job_name), exist_ok=True)
    # Get each suite within the job
    suites = client.list_suites(arn=job['arn'])['suites']
    for suite in suites:
        for test in client.list_tests(arn=suite['arn'])['tests']:
            # Get the artifacts
            for artifact_type in ['FILE', 'SCREENSHOT', 'LOG']:
                artifacts = client.list_artifacts(
                    type=artifact_type,
                    arn = test['arn']
                )['artifacts']
                for artifact in artifacts:
                    # We replace : because it has a special meaning in Windows & macos
                    path_to = os.path.join(save_path, job_name, suite['name'],
test['name'].replace(':', '_') )
                    os.makedirs(path_to, exist_ok=True)
                    filename =
artifact['type']+"_"+artifact['name']+"."+artifact['extension']
                    artifact_save_path = os.path.join(path_to, filename)
                    print("Downloading "+artifact_save_path)
                    with open(artifact_save_path, 'wb') as fn,
requests.get(artifact['url'],allow_redirects=True) as request:
                        fn.write(request.content)
                    #/for artifact in artifacts
                #/for artifact type in []
            #/ for test in ()[]
        #/ for suite in suites
    #/ for job in _[]
# done
print("Finished")

```



Résolution des erreurs liées à Device Farm

Dans cette section, vous trouverez des messages d'erreur et des procédures qui vous aideront à résoudre les problèmes courants liés à Device Farm.

Note

[Pour résoudre les problèmes liés à l'échec inattendu des tests Appium sur Device Farm, consultez notre guide sur les tests Appium côté client](#)

Rubriques

- [Résolution des problèmes liés aux tests d'applications Android dans AWS Device Farm](#)
- [Résolution des problèmes liés aux JUnit tests Java d'Appium dans AWS Device Farm](#)
- [Résolution des problèmes liés aux tests d'applications JUnit Web Appium Java dans AWS Device Farm](#)
- [Résolution des problèmes liés aux tests Appium Java TestNG dans AWS Device Farm](#)
- [Résolution des problèmes liés aux applications Web Appium Java TestNG dans AWS Device Farm](#)
- [Résolution des problèmes liés aux tests Appium Python dans AWS Device Farm](#)
- [Résolution des problèmes liés aux tests d'applications Web Appium Python dans AWS Device Farm](#)
- [Résolution des problèmes liés aux tests d'instrumentation dans AWS Device Farm](#)
- [Résolution des problèmes liés aux tests d'applications iOS dans AWS Device Farm](#)
- [XCTest Tests de résolution des problèmes dans AWS Device Farm](#)
- [Résolution des problèmes liés aux tests d' XCTest interface utilisateur dans AWS Device Farm](#)

Résolution des problèmes liés aux tests d'applications Android dans AWS Device Farm

La rubrique suivante répertorie les messages d'erreur qui s'affichent pendant le chargement des tests d'applications Android et recommande des solutions permettant de résoudre chaque erreur.

 Note

Les instructions ci-dessous sont basées sur Linux x86_64 et Mac.

ANDROID_APP_UNZIP_FAILED

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

 Warning

Impossible d'ouvrir votre application. Vérifiez que le fichier est valide et réessayez.

Assurez-vous que vous pouvez décompresser le package de l'application sans erreurs. Dans l'exemple suivant, le nom du package est app-debug.apk.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip app-debug.apk
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Un package d'application Android valide doit générer une sortie similaire à la suivante :

```
.
|-- AndroidManifest.xml
|-- classes.dex
|-- resources.arsc
|-- assets (directory)
|-- res (directory)
`-- META-INF (directory)
```

Pour de plus amples informations, veuillez consulter [Tests Android dans AWS Device Farm](#).

ANDROID_APP_AAPT_DEBUG_BADGING_FAILED

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible d'extraire des informations sur votre application. Vérifiez que l'application est valide en exécutant la commande `aapt debug badging <path to your test package>`, puis réessayez une fois que cette commande ne génère aucune erreur.

Au cours du processus de validation du téléchargement, AWS Device Farm analyse les informations issues de la sortie d'une `aapt debug badging <path to your package>` commande.

Assurez-vous que vous pouvez exécuter avec succès cette commande sur votre application Android. Dans l'exemple suivant, le nom du package est `app-debug.apk`.

- Copiez le package de votre application dans votre répertoire de travail, puis exécutez la commande :

```
$ aapt debug badging app-debug.apk
```

Un package d'application Android valide doit générer une sortie similaire à la suivante :

```
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'
```

Pour de plus amples informations, veuillez consulter [Tests Android dans AWS Device Farm](#).

ANDROID_APP_PACKAGE_NAME_VALUE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver la valeur du nom du package à l'intérieur de votre application. Veuillez vérifier que l'application est valide en exécutant la commande `aapt debug badging <path to your test package>`, puis réessayez après avoir trouvé la valeur du nom du package derrière le mot clé « package: name ».

Au cours du processus de validation du téléchargement, AWS Device Farm analyse la valeur du nom du package à partir de la sortie d'une `aapt debug badging <path to your package>` commande.

Assurez-vous que vous pouvez exécuter cette commande sur votre application Android et trouver la valeur du nom du package avec succès. Dans l'exemple suivant, le nom du package est `app-debug.apk`.

- Copiez le package de votre application dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ aapt debug badging app-debug.apk | grep "package: name="
```

Un package d'application Android valide doit générer une sortie similaire à la suivante :

```
package: name='com.amazon.aws.adf.android.referenceapp' versionCode='1'  
versionName='1.0' platformBuildVersionName='5.1.1-1819727'
```

Pour de plus amples informations, veuillez consulter [Tests Android dans AWS Device Farm](#).

ANDROID_APP_SDK_VERSION_VALUE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver la valeur de la version du kit SDK à l'intérieur de votre application. Veuillez vérifier que l'application est valide en exécutant la commande `aapt debug badging <path to your test package>`, puis réessayez après avoir trouvé la valeur du kit SDK derrière le mot clé `sdkVersion`.

Au cours du processus de validation du téléchargement, AWS Device Farm analyse la valeur de version du SDK à partir de la sortie d'une `aapt debug badging <path to your package>` commande.

Assurez-vous que vous pouvez exécuter cette commande sur votre application Android et trouver la valeur du nom du package avec succès. Dans l'exemple suivant, le nom du package est `app-debug.apk`.

- Copiez le package de votre application dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ aapt debug badging app-debug.apk | grep "sdkVersion"
```

Un package d'application Android valide doit générer une sortie similaire à la suivante :

```
sdkVersion:'9'
```

Pour de plus amples informations, veuillez consulter [Tests Android dans AWS Device Farm](#).

ANDROID_APP_AAPT_DUMP_XMLTREE_FAILED

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

 Warning

Nous n'avons pas trouvé le AndroidManifest fichier .xml valide dans votre application. Vérifiez que le package de test est valide en exécutant la commande `aapt dump xmltree <path to your test package> AndroidManifest.xml`, puis réessayez une fois que cette commande ne génère aucune erreur.

Au cours du processus de validation du téléchargement, AWS Device Farm analyse les informations de l'arbre d'analyse XML pour rechercher un fichier XML contenu dans le package à l'aide de la commande `aapt dump xmltree <path to your package> AndroidManifest.xml`

Assurez-vous que vous pouvez exécuter avec succès cette commande sur votre application Android. Dans l'exemple suivant, le nom du package est `app-debug.apk`.

- Copiez le package de votre application dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ aapt dump xmltree app-debug.apk. AndroidManifest.xml
```

Un package d'application Android valide doit générer une sortie similaire à la suivante :

```
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")
```

Pour de plus amples informations, veuillez consulter [Tests Android dans AWS Device Farm](#).

ANDROID_APP_DEVICE_ADMIN_PERMISSIONS

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Nous avons constaté que votre application nécessitait des autorisations d'administration d'appareil. Vérifiez que les autorisations ne sont pas requises par l'exécution de la commande `aapt dump xmltree <path to your test package> AndroidManifest.xml` et réessayez après vous être assuré que la sortie ne contient pas le mot clé `android.permission.BIND_DEVICE_ADMIN`.

Au cours du processus de validation du téléchargement, AWS Device Farm analyse les informations d'autorisation issues de l'arbre d'analyse XML pour un fichier XML contenu dans le package à l'aide de la commande `aapt dump xmltree <path to your package> AndroidManifest.xml`

Assurez-vous que votre application ne nécessite pas d'autorisation d'administration d'appareil. Dans l'exemple suivant, le nom du package est `app-debug.apk`.

- Copiez le package de votre application dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ aapt dump xmltree app-debug.apk AndroidManifest.xml
```

Vous devriez obtenir une sortie similaire à ce qui suit :

```
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")
.....
```

Si l'application Android est valide, la sortie ne doit pas contenir ce qui suit : A:
android:name(0x01010003)="android.permission.BIND_DEVICE_ADMIN" (Raw:
"android.permission.BIND_DEVICE_ADMIN").

Pour de plus amples informations, veuillez consulter [Tests Android dans AWS Device Farm](#).

Certaines fenêtres de mon application Android affichent un écran vide ou noir

Si vous testez une application Android et que vous remarquez que certaines fenêtres de l'application apparaissent avec un écran noir dans l'enregistrement vidéo de votre test par Device Farm, cela signifie que votre application utilise peut-être la FLAG_SECURE fonctionnalité Android. Ce drapeau (tel que décrit dans la [documentation officielle d'Android](#)) est utilisé pour empêcher l'enregistrement de certaines fenêtres d'une application par les outils d'enregistrement d'écran. Par conséquent, la fonction d'enregistrement d'écran de Device Farm (pour l'automatisation et les tests d'accès à distance) peut afficher un écran noir à la place de la fenêtre de votre application si celle-ci utilise cet indicateur.

Ce drapeau est souvent utilisé par les développeurs pour les pages de leur application contenant des informations sensibles telles que les pages de connexion. Si vous voyez un écran noir à la place de l'écran de votre application sur certaines pages, comme sa page de connexion, demandez à vos développeurs d'obtenir une version de l'application qui n'utilise pas cet indicateur pour les tests.

Notez également que Device Farm peut toujours interagir avec les fenêtres d'applications dotées de cet indicateur. Ainsi, si la page de connexion de votre application apparaît sous forme d'écran noir, vous pourrez peut-être toujours saisir vos informations d'identification pour vous connecter à l'application (et ainsi afficher les pages non bloquées par le FLAG_SECURE drapeau).

Résolution des problèmes liés aux JUnit tests Java d'Appium dans AWS Device Farm

La rubrique suivante répertorie les messages d'erreur qui se produisent lors du téléchargement des JUnit tests Java d'Appium et recommande des solutions pour résoudre chaque erreur.

Note

Les instructions ci-dessous sont basées sur Linux x86_64 et Mac.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible d'ouvrir votre fichier ZIP de test. Vérifiez que le fichier est valide et réessayez.

Assurez-vous que vous pouvez décompresser le package de test sans erreur. Dans l'exemple suivant, le nom du package est `zip-with-dependencies.zip`.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip zip-with-dependencies.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Un JUnit package Appium Java valide devrait produire une sortie comme celle-ci :

```
.
|- 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
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_JAVA_JUNIT_TEST_PACKAGE_DEPENDENCY_DIR_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver le répertoire dependency-jars à l'intérieur de votre package. Décompressez votre package de test, vérifiez que le répertoire dependency-jars s'y trouve, puis réessayez.

Dans l'exemple suivant, le nom du package est zip-with-dependencies.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip zip-with-dependencies.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le JUnit package Java Appium est valide, vous trouverez le *dependency-jars* répertoire dans le répertoire de travail :

```
.
|- 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
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DIR

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver un fichier JAR dans l'arborescence du répertoire dependency-jars. Décompressez votre package de test, puis ouvrez le répertoire dependency-jars, vérifiez que celui-ci contient au moins un fichier JAR, puis réessayez.

Dans l'exemple suivant, le nom du package est zip-with-dependencies.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip zip-with-dependencies.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```


Si le JUnit package Java Appium est valide, vous trouverez au moins un *jar* fichier dans le *dependency-jars* répertoire :

```
.
|- 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
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver un fichier *-tests.jar à l'intérieur de votre package. Décompressez votre package de test, vérifiez qu'au moins un fichier *-tests.jar s'y trouve, puis réessayez.

Dans l'exemple suivant, le nom du package est zip-with-dependencies.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip zip-with-dependencies.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le JUnit package Java d'Appium est valide, vous trouverez au moins un *jar* fichier comme *acme-android-appium-1.0-SNAPSHOT-tests.jar* dans notre exemple. Le nom du fichier peut être différent, mais il doit se terminer par *-tests.jar*.

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `-- log4j-1.2.14.jar
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JAR

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver un fichier de classe dans le fichier JAR de tests. Décompressez votre package de test, puis décompressez le fichier de tests JAR, vérifiez que celui-ci contient au moins un fichier de classe, puis réessayez.

Dans l'exemple suivant, le nom du package est zip-with-dependencies.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip zip-with-dependencies.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver au moins un fichier jar comme *acme-android-appium-1.0-SNAPSHOT-tests.jar* dans notre exemple. Le nom du fichier peut être différent, mais il doit se terminer par *-tests.jar*.

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `-- log4j-1.2.14.jar
```

3. Après avoir réussi à extraire les fichiers, vous devriez trouver au moins une classe dans le répertoire de travail arbre en exécutant la commande :

```
$ tree .
```

Elle doit générer une sortie comme suit :

```
.
|- 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
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_JAVA_JUNIT_TEST_PACKAGE_JUNIT_VERSION_VALUE_UNKNOWN

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Nous n'avons pas pu trouver de valeur de JUnit version. Décompressez votre package de test et ouvrez le répertoire dependency-jars, vérifiez que le fichier JUnit JAR se trouve dans le répertoire, puis réessayez.

Dans l'exemple suivant, le nom du package est zip-with-dependencies.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip zip-with-dependencies.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
tree .
```

La sortie doit se présenter comme suit :

```
.
|- 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
```

Si le JUnit package Java Appium est valide, vous trouverez le fichier de JUnit dépendance similaire au fichier jar *junit-4.10.jar* dans notre exemple. Le nom doit être composé du mot clé *junit* et de son numéro de version, qui dans cet exemple est 4.10.

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Nous avons constaté que la JUnit version était inférieure à la version minimale 4.10 que nous prenons en charge. Modifiez la JUnit version et réessayez.

Dans l'exemple suivant, le nom du package est zip-with-dependencies.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip zip-with-dependencies.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver un fichier de JUnit dépendance comme *junit-4.10.jar* dans notre exemple et son numéro de version, qui dans notre exemple est 4.10 :

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- junit-4.10.jar
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `- log4j-1.2.14.jar
```

Note

Vos tests risquent de ne pas s'exécuter correctement si la JUnit version spécifiée dans votre package de test est inférieure à la version minimale 4.10 que nous prenons en charge.

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

Résolution des problèmes liés aux tests d'applications JUnit Web Appium Java dans AWS Device Farm

La rubrique suivante répertorie les messages d'erreur qui se produisent lors du téléchargement des tests d'applications JUnit Web Appium Java et recommande des solutions pour résoudre chaque erreur. Pour plus d'informations sur l'utilisation d'Appium avec Device Farm, consultez [the section called "Tests Appium automatiques"](#)

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

 Warning

Impossible d'ouvrir votre fichier ZIP de test. Vérifiez que le fichier est valide et réessayez.

Assurez-vous que vous pouvez décompresser le package de test sans erreur. Dans l'exemple suivant, le nom du package est `zip-with-dependencies.zip`.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip zip-with-dependencies.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Un JUnit package Appium Java valide devrait produire une sortie comme celle-ci :

```
.
|- 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

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

 Warning

Impossible de trouver le répertoire `dependency-jars` à l'intérieur de votre package.
Décompressez votre package de test, vérifiez que le répertoire `dependency-jars` s'y trouve, puis réessayez.

Dans l'exemple suivant, le nom du package est `zip-with-dependencies.zip`.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip zip-with-dependencies.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le JUnit package Java Appium est valide, vous trouverez le *dependency-jars* répertoire dans le répertoire de travail :

```
.
|- 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

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

 Warning

Impossible de trouver un fichier JAR dans l'arborescence du répertoire `dependency-jars`. Décompressez votre package de test, puis ouvrez le répertoire `dependency-jars`, vérifiez que celui-ci contient au moins un fichier JAR, puis réessayez.

Dans l'exemple suivant, le nom du package est `zip-with-dependencies.zip`.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip zip-with-dependencies.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le JUnit package Java Appium est valide, vous trouverez au moins un *jar* fichier dans le *dependency-jars* répertoire :

```
.
|- 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

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

 Warning

Impossible de trouver un fichier *-tests.jar à l'intérieur de votre package. Décompressez votre package de test, vérifiez qu'au moins un fichier *-tests.jar s'y trouve, puis réessayez.

Dans l'exemple suivant, le nom du package est zip-with-dependencies.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip zip-with-dependencies.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le JUnit package Java d'Appium est valide, vous trouverez au moins un *jar* fichier comme *acme-android-appium-1.0-SNAPSHOT-tests.jar* dans notre exemple. Le nom du fichier peut être différent, mais il doit se terminer par *-tests.jar*.

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `- log4j-1.2.14.jar
```

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

 Warning

Impossible de trouver un fichier de classe dans le fichier JAR de tests. Décompressez votre package de test, puis décompressez le fichier de tests JAR, vérifiez que celui-ci contient au moins un fichier de classe, puis réessayez.

Dans l'exemple suivant, le nom du package est `zip-with-dependencies.zip`.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip zip-with-dependencies.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver au moins un fichier jar comme *acme-android-appium-1.0-SNAPSHOT-tests.jar* dans notre exemple. Le nom du fichier peut être différent, mais il doit se terminer par *-tests.jar*.

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `- log4j-1.2.14.jar
```

3. Après avoir réussi à extraire les fichiers, vous devriez trouver au moins une classe dans le répertoire de travail arbre en exécutant la commande :

```
$ tree .
```

Elle doit générer une sortie comme suit :

```
.
|- 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

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Nous n'avons pas pu trouver de valeur de JUnit version. Décompressez votre package de test et ouvrez le répertoire dependency-jars, vérifiez que le fichier JUnit JAR se trouve dans le répertoire, puis réessayez.

Dans l'exemple suivant, le nom du package est zip-with-dependencies.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip zip-with-dependencies.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
tree .
```

La sortie doit se présenter comme suit :

```
.
|- 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
```

Si le JUnit package Java Appium est valide, vous trouverez le fichier de JUnit dépendance similaire au fichier jar *junit-4.10.jar* dans notre exemple. Le nom doit être composé du mot clé *junit* et de son numéro de version, qui dans cet exemple est 4.10.

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Nous avons constaté que la JUnit version était inférieure à la version minimale 4.10 que nous prenons en charge. Modifiez la JUnit version et réessayez.

Dans l'exemple suivant, le nom du package est zip-with-dependencies.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip zip-with-dependencies.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver un fichier de JUnit dépendance comme *junit-4.10.jar* dans notre exemple et son numéro de version, qui dans notre exemple est 4.10 :

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- junit-4.10.jar
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `-- log4j-1.2.14.jar
```

Note

Vos tests risquent de ne pas s'exécuter correctement si la JUnit version spécifiée dans votre package de test est inférieure à la version minimale 4.10 que nous prenons en charge.

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

Résolution des problèmes liés aux tests Appium Java TestNG dans AWS Device Farm

La rubrique suivante répertorie les messages d'erreur qui s'affichent pendant le chargement des tests d'Appium Java TestNG et recommande des solutions permettant de résoudre chaque erreur.

Note

Les instructions ci-dessous sont basées sur Linux x86_64 et Mac.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible d'ouvrir votre fichier ZIP de test. Vérifiez que le fichier est valide et réessayez.

Assurez-vous que vous pouvez décompresser le package de test sans erreur. Dans l'exemple suivant, le nom du package est `zip-with-dependencies.zip`.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip zip-with-dependencies.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Un JUnit package Appium Java valide devrait produire une sortie comme celle-ci :

```
.
|- 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
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver le répertoire `dependency-jars` à l'intérieur de votre package. Décompressez votre package de test, vérifiez que le répertoire `dependency-jars` s'y trouve, puis réessayez.

Dans l'exemple suivant, le nom du package est `zip-with-dependencies.zip`.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip zip-with-dependencies.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le JUnit package Java Appium est valide, vous trouverez le *dependency-jars* répertoire dans le répertoire de travail.


```
.
|- 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
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DIR

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver un fichier JAR dans l'arborescence du répertoire dependency-jars. Décompressez votre package de test, puis ouvrez le répertoire dependency-jars, vérifiez que celui-ci contient au moins un fichier JAR, puis réessayez.

Dans l'exemple suivant, le nom du package est zip-with-dependencies.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip zip-with-dependencies.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le JUnit package Java Appium est valide, vous trouverez au moins un *jar* fichier dans le *dependency-jars* répertoire.

```
.
|- 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
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver un fichier *-tests.jar à l'intérieur de votre package. Décompressez votre package de test, vérifiez qu'au moins un fichier *-tests.jar s'y trouve, puis réessayez.

Dans l'exemple suivant, le nom du package est zip-with-dependencies.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip zip-with-dependencies.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le JUnit package Java d'Appium est valide, vous trouverez au moins un *jar* fichier comme *acme-android-appium-1.0-SNAPSHOT-tests.jar* dans notre exemple. Le nom du fichier peut être différent, mais il doit se terminer par *-tests.jar*.

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `-- log4j-1.2.14.jar
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver un fichier de classe dans le fichier JAR de tests. Décompressez votre package de test, puis décompressez le fichier de tests JAR, vérifiez que celui-ci contient au moins un fichier de classe, puis réessayez.

Dans l'exemple suivant, le nom du package est zip-with-dependencies.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip zip-with-dependencies.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver au moins un fichier jar comme *acme-android-appium-1.0-SNAPSHOT-tests.jar* dans notre exemple. Le nom du fichier peut être différent, mais il doit se terminer par *-tests.jar*.

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `-- log4j-1.2.14.jar
```

3. Pour extraire les fichiers du fichier jar, vous pouvez exécuter la commande suivante :

```
$ jar xf acme-android-appium-1.0-SNAPSHOT-tests.jar
```

4. Une fois que vous avez réussi à extraire les fichiers, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver au moins une classe dans l'arborescence du répertoire de travail :

```
.
|- 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
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

Résolution des problèmes liés aux applications Web Appium Java TestNG dans AWS Device Farm

La rubrique suivante répertorie les messages d'erreur qui s'affichent pendant le chargement des tests d'application Web d'Appium Java TestNG et recommande des solutions permettant de résoudre chaque erreur.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible d'ouvrir votre fichier ZIP de test. Vérifiez que le fichier est valide et réessayez.

Assurez-vous que vous pouvez décompresser le package de test sans erreur. Dans l'exemple suivant, le nom du package est zip-with-dependencies.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip zip-with-dependencies.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Un JUnit package Appium Java valide devrait produire une sortie comme celle-ci :

```
.
|- 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
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver le répertoire dependency-jars à l'intérieur de votre package.
Décompressez votre package de test, vérifiez que le répertoire dependency-jars s'y trouve,
puis réessayez.

Dans l'exemple suivant, le nom du package est zip-with-dependencies.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip zip-with-dependencies.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le JUnit package Java Appium est valide, vous trouverez le *dependency-jars* répertoire dans le répertoire de travail.

```
.
|- 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
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DIR

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver un fichier JAR dans l'arborescence du répertoire dependency-jars. Décompressez votre package de test, puis ouvrez le répertoire dependency-jars, vérifiez que celui-ci contient au moins un fichier JAR, puis réessayez.

Dans l'exemple suivant, le nom du package est zip-with-dependencies.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip zip-with-dependencies.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le JUnit package Java Appium est valide, vous trouverez au moins un *jar* fichier dans le *dependency-jars* répertoire.

```
.
|- 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
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver un fichier *-tests.jar à l'intérieur de votre package. Décompressez votre package de test, vérifiez qu'au moins un fichier *-tests.jar s'y trouve, puis réessayez.

Dans l'exemple suivant, le nom du package est zip-with-dependencies.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip zip-with-dependencies.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le JUnit package Java d'Appium est valide, vous trouverez au moins un *jar* fichier comme *acme-android-appium-1.0-SNAPSHOT-tests.jar* dans notre exemple. Le nom du fichier peut être différent, mais il doit se terminer par *-tests.jar*.

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `-- log4j-1.2.14.jar
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_T

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

⚠ Warning

Impossible de trouver un fichier de classe dans le fichier JAR de tests. Décompressez votre package de test, puis décompressez le fichier de tests JAR, vérifiez que celui-ci contient au moins un fichier de classe, puis réessayez.

Dans l'exemple suivant, le nom du package est `zip-with-dependencies.zip`.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip zip-with-dependencies.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver au moins un fichier jar comme *acme-android-appium-1.0-SNAPSHOT-tests.jar* dans notre exemple. Le nom du fichier peut être différent, mais il doit se terminer par *-tests.jar*.

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `- log4j-1.2.14.jar
```

3. Pour extraire les fichiers du fichier jar, vous pouvez exécuter la commande suivante :

```
$ jar xf acme-android-appium-1.0-SNAPSHOT-tests.jar
```

4. Une fois que vous avez réussi à extraire les fichiers, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver au moins une classe dans l'arborescence du répertoire de travail :

```
.
|- 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
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

Résolution des problèmes liés aux tests Appium Python dans AWS Device Farm

La rubrique suivante répertorie les messages d'erreur qui s'affichent pendant le chargement des tests Appium écrits en Python et propose des solutions pour résoudre chaque erreur.

APPIUM_PYTHON_TEST_PACKAGE_UNZIP_FAILED

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

 Warning

Impossible d'ouvrir votre fichier ZIP de test Appium. Vérifiez que le fichier est valide et réessayez.

Assurez-vous que vous pouvez décompresser le package de test sans erreur. Dans l'exemple suivant, le nom du package est `test_bundle.zip`.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip test_bundle.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Un package Appium Python valide doit générer une sortie similaire à la suivante :

```
.
|-- 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
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

 Warning

Impossible de trouver un fichier wheel de dépendance dans l'arborescence du répertoire wheelhouse. Décompressez votre package de test, puis ouvrez le répertoire wheelhouse, vérifiez que celui-ci contient au moins un fichier wheel, puis réessayez.

Assurez-vous que vous pouvez décompresser le package de test sans erreur. Dans l'exemple suivant, le nom du package est test_bundle.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip test_bundle.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le package Appium Python est valide, vous trouverez au moins un fichier **.whl** dépendant, comme les fichiers surlignés, dans le **wheelhouse** répertoire.

```
.
|-- 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
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_PYTHON_TEST_PACKAGE_INVALID_PLATFORM

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Au moins un fichier wheel spécifie une plateforme que nous ne prenons pas en charge. Décompressez votre package de test, puis ouvrez le répertoire wheelhouse, vérifiez que les noms de fichiers wheel se terminent par `-any.whl` ou `-linux_x86_64.whl` et réessayez.

Assurez-vous que vous pouvez décompresser le package de test sans erreur. Dans l'exemple suivant, le nom du package est `test_bundle.zip`.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip test_bundle.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le package Appium Python est valide, vous trouverez au moins un fichier `.whl` dépendant, comme les fichiers surlignés, dans le `wheelhouse` répertoire. Le nom du fichier peut être différent, mais il doit se terminer par `-any.whl` ou `-linux_x86_64.whl`, qui indique la plateforme. Toutes les plateformes, comme windows, ne sont pas prises en charge.

```
.
|-- 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
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver le répertoire de tests à l'intérieur de votre package. Décompressez votre package de test, vérifiez que le répertoire de tests s'y trouve, puis réessayez.

Assurez-vous que vous pouvez décompresser le package de test sans erreur. Dans l'exemple suivant, le nom du package est test_bundle.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip test_bundle.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le package Python d'Appium est valide, vous trouverez le *tests* répertoire dans le répertoire de travail.

```
.
|-- 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
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver un fichier de test valide dans l'arborescence du répertoire de tests. Décompressez votre package de test, ouvrez le répertoire de tests, vérifiez que le nom d'au moins un fichier commence ou finit par le mot clé « test », puis réessayez.

Assurez-vous que vous pouvez décompresser le package de test sans erreur. Dans l'exemple suivant, le nom du package est `test_bundle.zip`.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip test_bundle.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le package Python d'Appium est valide, vous trouverez le `tests` répertoire dans le répertoire de travail. Le nom du fichier peut être différent, mais il doit commencer par `test_` ou se terminer par `_test.py`.

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
```



```
|      `-- test_unittest.py
|-- wheelhouse (directory)
|   |-- Appium_Python_Client-0.20-cp27-none-any.whl
|   |-- py-1.4.31-py2.py3-none-any.whl
|   |-- pytest-2.9.0-py2.py3-none-any.whl
|   |-- selenium-2.52.0-cp27-none-any.whl
|   `-- wheel-0.26.0-py2.py3-none-any.whl
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver le fichier requirements.txt à l'intérieur de votre package. Décompressez votre package de test, vérifiez que le fichier requirements.txt s'y trouve, puis réessayez.

Assurez-vous que vous pouvez décompresser le package de test sans erreur. Dans l'exemple suivant, le nom du package est test_bundle.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip test_bundle.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le package Python d'Appium est valide, vous trouverez le *requirements.txt* fichier dans le répertoire de travail.

```
.
|-- 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
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

La version de pytest que vous utilisez est antérieure à la version 2.8.0 minimale que nous prenons en charge. Modifiez la version de pytest du fichier requirements.txt et réessayez.

Assurez-vous que vous pouvez décompresser le package de test sans erreur. Dans l'exemple suivant, le nom du package est test_bundle.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip test_bundle.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver le *requirements.txt* fichier dans le répertoire de travail.

```
.
```

```
|-- 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. Pour connaître la version de pytest, exécutez la commande suivante :

```
$ grep "pytest" requirements.txt
```

Vous devriez obtenir une sortie similaire à ce qui suit :

```
pytest==2.9.0
```

Elle montre la version de pytest, qui est 2.9.0 dans cet exemple. Si le package Appium Python est valide, la version de pytest doit être supérieure ou égale à 2.8.0.

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS_FAIL

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible d'installer les fichiers wheel de dépendance. Décompressez votre package de test, ouvrez le fichier requirements.txt et le répertoire wheelhouse, vérifiez que les fichiers wheel de dépendance spécifiés dans le fichier requirements.txt correspondent exactement aux fichiers wheel de dépendance situés dans le répertoire wheelhouse et réessayez.

Nous vous recommandons vivement de configurer [virtualenv Python](#) pour les tests de création de package. Voici un exemple de flux de création d'un environnement virtuel à l'aide de virtualenv Python et de son activation :

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

Assurez-vous que vous pouvez décompresser le package de test sans erreur. Dans l'exemple suivant, le nom du package est test_bundle.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip test_bundle.zip
```

2. Pour tester l'installation des fichiers wheel, vous pouvez exécuter la commande suivante :

```
$ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./requirements.txt
```

Un package Appium Python valide doit générer une sortie similaire à la suivante :

```
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. Pour désactiver l'environnement virtuel, exécutez la commande suivante :

```
$ deactivate
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Echec de la collecte de tests dans le répertoire de tests. Décompressez votre package de test, vérifiez qu'il est valide en exécutant la commande `py.test --collect-only <path to your tests directory>`, puis réessayez une fois que cette commande ne génère aucune erreur.

Nous vous recommandons vivement de configurer [virtualenv Python](#) pour les tests de création de package. Voici un exemple de flux de création d'un environnement virtuel à l'aide de virtualenv Python et de son activation :

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

Assurez-vous que vous pouvez décompresser le package de test sans erreur. Dans l'exemple suivant, le nom du package est `test_bundle.zip`.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip test_bundle.zip
```

2. Pour installer des fichiers wheel, vous pouvez exécuter la commande suivante :

```
$ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./requirements.txt
```

3. Pour collecter des tests, vous pouvez exécuter la commande suivante :

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

Un package Appium Python valide doit générer une sortie similaire à la suivante :

```
===== test session starts =====
platform darwin -- Python 2.7.11, pytest-2.9.0, py-1.4.31, pluggy-0.3.1
rootdir: /Users/zhenan/Desktop/Ios/tests, inifile:
collected 1 items
<Module 'test_unittest.py'>
  <UnitTestCase 'DeviceFarmAppiumWebTests'>
    <TestCaseFunction 'test_devicefarm'>

===== no tests ran in 0.11 seconds =====
```

4. Pour désactiver l'environnement virtuel, exécutez la commande suivante :

```
$ deactivate
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEELS_INSUFFICIENT

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Nous n'avons pas trouvé suffisamment de dépendances entre les roues dans le répertoire des roues. Décompressez votre package de test, puis ouvrez le répertoire Wheelhouse. Vérifiez que toutes les dépendances des roues sont spécifiées dans le fichier requirements.txt.

Assurez-vous que vous pouvez décompresser le package de test sans erreur. Dans l'exemple suivant, le nom du package est test_bundle.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip test_bundle.zip
```

2. Vérifiez la longueur du `requirements.txt` fichier ainsi que le nombre de fichiers `.whl` dépendants dans le répertoire de la timonerie :

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

Si le nombre de fichiers `.whl` dépendants est inférieur au nombre de lignes non vides de votre `requirements.txt` fichier, vous devez vous assurer de ce qui suit :

- Un fichier `.whl` dépendant correspond à chaque ligne du `requirements.txt` fichier.
- Aucune autre ligne du `requirements.txt` fichier ne contient d'informations autres que les noms des packages de dépendance.
- Aucun nom de dépendance n'est dupliqué sur plusieurs lignes du `requirements.txt` fichier, de sorte que deux lignes du fichier peuvent correspondre à un fichier `.whl` dépendant.

AWS Device Farm ne prend pas en charge les lignes du `requirements.txt` fichier qui ne correspondent pas directement aux packages de dépendances, telles que les lignes qui spécifient des options globales pour la `pip install` commande. Voir [Format de fichier d'exigences](#) pour une liste des options globales.

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

Résolution des problèmes liés aux tests d'applications Web Appium Python dans AWS Device Farm

La rubrique suivante répertorie les messages d'erreur qui s'affichent pendant le chargement des tests d'application Web d'Appium Python et recommande des solutions permettant de résoudre chaque erreur.

APPIUM_WEB_PYTHON_TEST_PACKAGE_UNZIP_FAILED

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible d'ouvrir votre fichier ZIP de test Appium. Vérifiez que le fichier est valide et réessayez.

Assurez-vous que vous pouvez décompresser le package de test sans erreur. Dans l'exemple suivant, le nom du package est `test_bundle.zip`.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip test_bundle.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Un package Appium Python valide doit générer une sortie similaire à la suivante :

```
.
|-- 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
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_WEB_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver un fichier wheel de dépendance dans l'arborescence du répertoire wheelhouse. Décompressez votre package de test, puis ouvrez le répertoire wheelhouse, vérifiez que celui-ci contient au moins un fichier wheel, puis réessayez.

Assurez-vous que vous pouvez décompresser le package de test sans erreur. Dans l'exemple suivant, le nom du package est test_bundle.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip test_bundle.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le package Appium Python est valide, vous trouverez au moins un fichier **.whl** dépendant, comme les fichiers surlignés, dans le **wheelhouse** répertoire.

```
.
|-- 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
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PLATFORM

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Au moins un fichier wheel spécifie une plateforme que nous ne prenons pas en charge. Décompressez votre package de test, puis ouvrez le répertoire wheelhouse, vérifiez que les noms de fichiers wheel se terminent par `-any.whl` ou `-linux_x86_64.whl` et réessayez.

Assurez-vous que vous pouvez décompresser le package de test sans erreur. Dans l'exemple suivant, le nom du package est `test_bundle.zip`.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip test_bundle.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le package Appium Python est valide, vous trouverez au moins un fichier `.whl` dépendant, comme les fichiers surlignés, dans le `wheelhouse` répertoire. Le nom du fichier peut être différent, mais il doit se terminer par `-any.whl` ou `-linux_x86_64.whl`, qui indique la plateforme. Toutes les plateformes, comme windows, ne sont pas prises en charge.

```
.
|-- 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
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_WEB_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver le répertoire de tests à l'intérieur de votre package. Décompressez votre package de test, vérifiez que le répertoire de tests s'y trouve, puis réessayez.

Assurez-vous que vous pouvez décompresser le package de test sans erreur. Dans l'exemple suivant, le nom du package est test_bundle.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip test_bundle.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le package Python d'Appium est valide, vous trouverez le *tests* répertoire dans le répertoire de travail.

```
.  
|-- 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
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver un fichier de test valide dans l'arborescence du répertoire de tests. Décompressez votre package de test, ouvrez le répertoire de tests, vérifiez que le nom d'au moins un fichier commence ou finit par le mot clé « test », puis réessayez.

Assurez-vous que vous pouvez décompresser le package de test sans erreur. Dans l'exemple suivant, le nom du package est `test_bundle.zip`.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip test_bundle.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le package Python d'Appium est valide, vous trouverez le `tests` répertoire dans le répertoire de travail. Le nom du fichier peut être différent, mais il doit commencer par `test_` ou se terminer par `_test.py`.

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|   |-- test_unittest.py
|-- wheelhouse (directory)
|   |-- Appium_Python_Client-0.20-cp27-none-any.whl
|   |-- py-1.4.31-py2.py3-none-any.whl
|   |-- pytest-2.9.0-py2.py3-none-any.whl
|   |-- selenium-2.52.0-cp27-none-any.whl
|   |-- wheel-0.26.0-py2.py3-none-any.whl
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_WEB_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver le fichier requirements.txt à l'intérieur de votre package. Décompressez votre package de test, vérifiez que le fichier requirements.txt s'y trouve, puis réessayez.

Assurez-vous que vous pouvez décompresser le package de test sans erreur. Dans l'exemple suivant, le nom du package est test_bundle.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip test_bundle.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le package Python d'Appium est valide, vous trouverez le *requirements.txt* fichier dans le répertoire de travail.

```
.
|-- 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
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

La version de pytest que vous utilisez est antérieure à la version 2.8.0 minimale que nous prenons en charge. Modifiez la version de pytest du fichier requirements.txt et réessayez.

Assurez-vous que vous pouvez décompresser le package de test sans erreur. Dans l'exemple suivant, le nom du package est test_bundle.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip test_bundle.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver le *requirements.txt* fichier dans le répertoire de travail.

```
.
|-- 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. Pour connaître la version de pytest, exécutez la commande suivante :

```
$ grep "pytest" requirements.txt
```

Vous devriez obtenir une sortie similaire à ce qui suit :

```
pytest==2.9.0
```

Elle montre la version de pytest, qui est 2.9.0 dans cet exemple. Si le package Appium Python est valide, la version de pytest doit être supérieure ou égale à 2.8.0.

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_WEB_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS_FAILED

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible d'installer les fichiers wheel de dépendance. Décompressez votre package de test, ouvrez le fichier requirements.txt et le répertoire wheelhouse, vérifiez que les fichiers

wheel de dépendance spécifiés dans le fichier requirements.txt correspondent exactement aux fichiers wheel de dépendance situés dans le répertoire wheelhouse et réessayez.

Nous vous recommandons vivement de configurer [virtualenv Python](#) pour les tests de création de package. Voici un exemple de flux de création d'un environnement virtuel à l'aide de virtualenv Python et de son activation :

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

Assurez-vous que vous pouvez décompresser le package de test sans erreur. Dans l'exemple suivant, le nom du package est test_bundle.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip test_bundle.zip
```

2. Pour tester l'installation des fichiers wheel, vous pouvez exécuter la commande suivante :

```
$ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./requirements.txt
```

Un package Appium Python valide doit générer une sortie similaire à la suivante :

```
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. Pour désactiver l'environnement virtuel, exécutez la commande suivante :


```
$ deactivate
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

APPIUM_WEB_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Echec de la collecte de tests dans le répertoire de tests. Décompressez votre package de test, vérifiez qu'il est valide en exécutant la commande « `py.test --collect-only <chemin de votre répertoire tests>` », puis réessayez une fois que cette commande ne génère aucune erreur.

Nous vous recommandons vivement de configurer [virtualenv Python](#) pour les tests de création de package. Voici un exemple de flux de création d'un environnement virtuel à l'aide de virtualenv Python et de son activation :

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

Assurez-vous que vous pouvez décompresser le package de test sans erreur. Dans l'exemple suivant, le nom du package est `test_bundle.zip`.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip test_bundle.zip
```

2. Pour installer des fichiers wheel, vous pouvez exécuter la commande suivante :

```
$ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./requirements.txt
```

3. Pour collecter des tests, vous pouvez exécuter la commande suivante :

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

Un package Appium Python valide doit générer une sortie similaire à la suivante :

```
===== test session starts =====
platform darwin -- Python 2.7.11, pytest-2.9.0, py-1.4.31, pluggy-0.3.1
rootdir: /Users/zhenan/Desktop/Ios/tests, inifile:
collected 1 items
<Module 'test_unittest.py'>
  <UnitTestCase 'DeviceFarmAppiumWebTests'>
    <TestCaseFunction 'test_devicefarm'>

===== no tests ran in 0.11 seconds =====
```

4. Pour désactiver l'environnement virtuel, exécutez la commande suivante :

```
$ deactivate
```

Pour de plus amples informations, veuillez consulter [Exécuter automatiquement des tests Appium dans Device Farm](#).

Résolution des problèmes liés aux tests d'instrumentation dans AWS Device Farm

La rubrique suivante répertorie les messages d'erreur qui s'affichent pendant le chargement des tests Instrumentation et recommande des solutions permettant de résoudre chaque erreur.

Note

Pour connaître les points importants à prendre en compte lors de l'utilisation des tests d'instrumentation dans AWS Device Farm, consultez [Instrumentation pour Android et AWS Device Farm](#).

INSTRUMENTATION_TEST_PACKAGE_UNZIP_FAILED

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

```
Warning: We could not open your test APK file. Please verify that the file is valid and try again.
```

Assurez-vous que vous pouvez décompresser le package de test sans erreur. Dans l'exemple suivant, le nom du package est `app-debug-androidTest-unaligned.apk`.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip app-debug-androidTest-unaligned.apk
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Un package d'application Instrumentation valide doit générer une sortie similaire à la suivante :

```
.
|-- AndroidManifest.xml
|-- classes.dex
|-- resources.arsc
|-- LICENSE-junit.txt
|-- junit (directory)
`-- META-INF (directory)
```

Pour de plus amples informations, veuillez consulter [Instrumentation pour Android et AWS Device Farm](#).

INSTRUMENTATION_TEST_PACKAGE_AAPT_DEBUG_BADGING_FAILED

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

```
We could not extract information about your test package. Please verify that the
```

```
test package is valid by running the command "aapt debug badging <path to your
test package>", and try again after the command does not print any error.
```

Pendant le processus de validation du téléchargement, Device Farm analyse les informations issues de la `aapt debug badging <path to your package>` commande.

Assurez-vous que vous pouvez exécuter avec succès cette commande sur votre package de test Instrumentation.

Dans l'exemple suivant, le nom du package est `app-debug-androidTest-unaligned.apk`.

- Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ aapt debug badging app-debug-androidTest-unaligned.apk
```

Un package d'application Instrumentation valide doit générer une sortie similaire à la suivante :

```
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'
```

Pour de plus amples informations, veuillez consulter [Instrumentation pour Android et AWS Device Farm](#).

INSTRUMENTATION_TEST_PACKAGE_INSTRUMENTATION_RUNNER_VALU

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

```
We could not find the instrumentation runner value in the AndroidManifest.xml.  
Please verify the test package is valid by running the command "aapt dump xmltree  
<path to  
your test package> AndroidManifest.xml", and try again after finding the  
instrumentation  
runner value behind the keyword "instrumentation."
```

Au cours du processus de validation du téléchargement, Device Farm analyse la valeur du lanceur d'instrumentation à partir de l'arbre d'analyse XML pour un fichier XML contenu dans le package. Vous pouvez utiliser la commande suivante : `aapt dump xmltree <path to your package> AndroidManifest.xml`.

Assurez-vous que vous pouvez exécuter cette commande sur votre package de test Instrumentation et trouver la valeur d'instrumentation avec succès.

Dans l'exemple suivant, le nom du package est `app-debug-androidTest-unaligned.apk`.

- Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ aapt dump xmltree app-debug-androidTest-unaligned.apk AndroidManifest.xml | grep  
-A5 "instrumentation"
```

Un package d'application Instrumentation valide doit générer une sortie similaire à la suivante :

```
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
```

Pour de plus amples informations, veuillez consulter [Instrumentation pour Android et AWS Device Farm](#).

INSTRUMENTATION_TEST_PACKAGE_AAPT_DUMP_XMLTREE_FAILED

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

```
We could not find the valid AndroidManifest.xml in your test package. Please
    verify that the test package is valid by running the command "aapt dump xmltree
    <path to
    your test package> AndroidManifest.xml", and try again after the command does not
    print any
    error.
```

Pendant le processus de validation du téléchargement, Device Farm analyse les informations de l'arbre d'analyse XML pour un fichier XML contenu dans le package à l'aide de la commande suivante : `aapt dump xmltree <path to your package> AndroidManifest.xml`

Assurez-vous que vous pouvez exécuter avec succès cette commande sur votre package de test d'instrumentation.

Dans l'exemple suivant, le nom du package est `app-debug-androidTest-unaligned.apk`.

- Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ aapt dump xmltree app-debug-androidTest-unaligned.apk AndroidManifest.xml
```

Un package d'application Instrumentation valide doit générer une sortie similaire à la suivante :

```
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")

```

Pour de plus amples informations, veuillez consulter [Instrumentation pour Android et AWS Device Farm](#).

INSTRUMENTATION_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

```

We could not find the package name in your test package. Please verify that the
test package is valid by running the command "aapt debug badging <path to your
test
package>", and try again after finding the package name value behind the keyword
"package:
name."

```

Pendant le processus de validation du téléchargement, Device Farm analyse la valeur du nom du package à partir de la sortie de la commande suivante : `aapt debug badging <path to your package>`.

Assurez-vous que vous pouvez exécuter cette commande sur votre package de test Instrumentation et trouver la valeur du nom du package avec succès.

Dans l'exemple suivant, le nom du package est `app-debug-androidTest-unaligned.apk`.

- Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ aapt debug badging app-debug-androidTest-unaligned.apk | grep "package: name="
```

Un package d'application Instrumentation valide doit générer une sortie similaire à la suivante :

```
package: name='com.amazon.aws.adf.android.referenceapp.test' versionCode=''  
versionName='' platformBuildVersionName='5.1.1-1819727'
```

Pour de plus amples informations, veuillez consulter [Instrumentation pour Android et AWS Device Farm](#).

Résolution des problèmes liés aux tests d'applications iOS dans AWS Device Farm

La rubrique suivante répertorie les messages d'erreur qui s'affichent pendant le chargement des tests d'applications iOS et recommande des solutions permettant de résoudre chaque erreur.

Note

Les instructions ci-dessous sont basées sur Linux x86_64 et Mac.

IOS_APP_UNZIP_FAILED

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible d'ouvrir votre application. Vérifiez que le fichier est valide et réessayez.

Assurez-vous que vous pouvez décompresser le package de l'application sans erreurs. Dans l'exemple suivant, le nom du package est AWSDeviceFarm iOSReference App.ipa.

1. Copiez le package de votre application dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Un package d'application iOS valide doit générer une sortie similaire à la suivante :

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

Pour de plus amples informations, veuillez consulter [Tests iOS dans AWS Device Farm](#).

IOS_APP_PAYLOAD_DIR_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver le répertoire Payload à l'intérieur de votre application. Décompressez votre application, vérifiez que le répertoire Payload s'y trouve, puis réessayez.

Dans l'exemple suivant, le nom du package est AWSDeviceFarm iOSReference App.ipa.

1. Copiez le package de votre application dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le package de l'application iOS est valide, vous trouverez le *Payload* répertoire dans le répertoire de travail.

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

Pour de plus amples informations, veuillez consulter [Tests iOS dans AWS Device Farm](#).

IOS_APP_APP_DIR_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver le répertoire .app à l'intérieur de votre répertoire Payload.
Décompressez votre application, puis ouvrez le répertoire Payload, vérifiez que le répertoire .app est à l'intérieur du répertoire, puis réessayez.

Dans l'exemple suivant, le nom du package est AWSDeviceFarm iOSReference App.ipa.

1. Copiez le package de votre application dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le package de l'application iOS est valide, vous trouverez un **.app** répertoire comme **AWSDeviceFarmiOSReferenceApp.app** dans notre exemple à l'intérieur du **Payload** répertoire.

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

Pour de plus amples informations, veuillez consulter [Tests iOS dans AWS Device Farm](#).

IOS_APP_PLIST_FILE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver le fichier Info.plist à l'intérieur du répertoire .app. Décompressez votre application, puis ouvrez le répertoire .app, vérifiez que le fichier Info.plist est à l'intérieur du répertoire, puis réessayez.

Dans l'exemple suivant, le nom du package est AWSDeviceFarm iOSReference App.ipa.

1. Copiez le package de votre application dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le package de l'application iOS est valide, vous trouverez le **Info.plist** fichier dans le **.app** répertoire comme **AWSDeviceFarmiOSReferenceApp.app** dans notre exemple.

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

Pour de plus amples informations, veuillez consulter [Tests iOS dans AWS Device Farm](#).

IOS_APP_CPU_ARCHITECTURE_VALUE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver la valeur de l'architecture d'UC à l'intérieur du fichier Info.plist. Décompressez votre application, puis ouvrez le fichier Info.plist dans le répertoire .app, vérifiez que la clé « UIRequired DeviceCapabilities » est spécifiée et réessayez.

Dans l'exemple suivant, le nom du package est AWSDeviceFarm iOSReference App.ipa.

1. Copiez le package de votre application dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver le *Info.plist* fichier dans un *.app* répertoire comme *AWSDeviceFarmiOSReferenceApp.app* dans notre exemple :

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

```
`-- (any other files)
```

3. Pour rechercher la valeur de l'architecture d'UC, vous pouvez ouvrir Info.plist à l'aide de Xcode ou de Python.

Pour Python, vous pouvez installer le module biplist en exécutant la commande suivante :

```
$ pip install biplist
```

4. Ensuite, ouvrez Python et exécutez la commande suivante :

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['UIRequiredDeviceCapabilities']
```

Un package d'application iOS valide doit générer une sortie similaire à la suivante :

```
['armv7']
```

Pour de plus amples informations, veuillez consulter [Tests iOS dans AWS Device Farm](#).

IOS_APP_PLATFORM_VALUE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver la valeur de la plateforme à l'intérieur du fichier Info.plist.
Décompressez votre application, puis ouvrez le fichier Info.plist dans le répertoire .app, vérifiez que la clé « CFBundle SupportedPlatforms » est spécifiée et réessayez.

Dans l'exemple suivant, le nom du package est AWSDeviceFarmiOSReferenceApp.ipa.

1. Copiez le package de votre application dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver le *Info.plist* fichier dans un *.app* répertoire comme *AWSDeviceFarmiOSReferenceApp.app* dans notre exemple :

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

3. Pour rechercher la valeur de la plateforme, vous pouvez ouvrir Info.plist à l'aide de Xcode ou de Python.

Pour Python, vous pouvez installer le module biplist en exécutant la commande suivante :

```
$ pip install biplist
```

4. Ensuite, ouvrez Python et exécutez la commande suivante :

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/Info.plist')
print info_plist['CFBundleSupportedPlatforms']
```

Un package d'application iOS valide doit générer une sortie similaire à la suivante :

```
['iPhoneOS']
```

Pour de plus amples informations, veuillez consulter [Tests iOS dans AWS Device Farm](#).

IOS_APP_WRONG_PLATFORM_DEVICE_VALUE

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

⚠ Warning

Nous avons découvert que la valeur de l'appareil de la plateforme était incorrecte dans le fichier Info.plist. Décompressez votre application, puis ouvrez le fichier Info.plist dans le répertoire .app, vérifiez que la valeur de la clé « » CFBundle SupportedPlatforms ne contient pas le mot-clé « simulator », puis réessayez.

Dans l'exemple suivant, le nom du package est AWSDeviceFarmiOSReference App.ipa.

1. Copiez le package de votre application dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver le *Info.plist* fichier dans un *.app* répertoire comme *AWSDeviceFarmiOSReferenceApp.app* dans notre exemple :

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

3. Pour rechercher la valeur de la plateforme, vous pouvez ouvrir Info.plist à l'aide de Xcode ou de Python.

Pour Python, vous pouvez installer le module biplist en exécutant la commande suivante :

```
$ pip install biplist
```

4. Ensuite, ouvrez Python et exécutez la commande suivante :

```
import biplist
```

```
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/Info.plist')
print info_plist['CFBundleSupportedPlatforms']
```

Un package d'application iOS valide doit générer une sortie similaire à la suivante :

```
['iPhoneOS']
```

Si l'application iOS est valide, la valeur ne doit pas contenir le mot clé `simulator`.

Pour de plus amples informations, veuillez consulter [Tests iOS dans AWS Device Farm](#).

IOS_APP_FORM_FACTOR_VALUE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver la valeur du facteur de format à l'intérieur du fichier Info.plist. Décompressez votre application, puis ouvrez le fichier Info.plist dans le répertoire .app, vérifiez que la clé « UIDevice Family » est spécifiée et réessayez.

Dans l'exemple suivant, le nom du package est AWSDeviceFarm iOSReference App.ipa.

1. Copiez le package de votre application dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver le *Info.plist* fichier dans un *.app* répertoire comme *AWSDeviceFarmiOSReferenceApp.app* dans notre exemple :

```
.
```



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

3. Pour rechercher la valeur du facteur de formulaire, vous pouvez ouvrir Info.plist à l'aide de Xcode ou de Python.

Pour Python, vous pouvez installer le module biplist en exécutant la commande suivante :

```
$ pip install biplist
```

4. Ensuite, ouvrez Python et exécutez la commande suivante :

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['UIDeviceFamily']
```

Un package d'application iOS valide doit générer une sortie similaire à la suivante :

```
[1, 2]
```

Pour de plus amples informations, veuillez consulter [Tests iOS dans AWS Device Farm](#).

IOS_APP_PACKAGE_NAME_VALUE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver la valeur du nom du package à l'intérieur du fichier Info.plist. Décompressez votre application, puis ouvrez le fichier Info.plist dans le répertoire .app, vérifiez que la clé « CFBUNDLE_IDENTIFIER » est spécifiée et réessayez.

Dans l'exemple suivant, le nom du package est AWSDeviceFarm iOSReference App.ipa.

1. Copiez le package de votre application dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver le *Info.plist* fichier dans un *.app* répertoire comme *AWSDeviceFarmiOSReferenceApp.app* dans notre exemple :

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

3. Pour rechercher la valeur du nom du package, vous pouvez ouvrir Info.plist à l'aide de Xcode ou de Python.

Pour Python, vous pouvez installer le module biplist en exécutant la commande suivante :

```
$ pip install biplist
```

4. Ensuite, ouvrez Python et exécutez la commande suivante :

```
import biplist  
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/  
Info.plist')  
print info_plist['CFBundleIdentifier']
```

Un package d'application iOS valide doit générer une sortie similaire à la suivante :

```
Amazon.AWSDeviceFarmiOSReferenceApp
```

Pour de plus amples informations, veuillez consulter [Tests iOS dans AWS Device Farm](#).

IOS_APP_EXECUTABLE_VALUE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver la valeur du fichier exécutable à l'intérieur du fichier Info.plist. Décompressez votre application, puis ouvrez le fichier Info.plist dans le répertoire .app, vérifiez que la clé « CFBundle Executable » est spécifiée et réessayez.

Dans l'exemple suivant, le nom du package est AWSDeviceFarmiOSReferenceApp.ipa.

1. Copiez le package de votre application dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver le *Info.plist* fichier dans un *.app* répertoire comme *AWSDeviceFarmiOSReferenceApp.app* dans notre exemple :

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

3. Pour rechercher la valeur du fichier exécutable, vous pouvez ouvrir Info.plist à l'aide de Xcode ou de Python.

Pour Python, vous pouvez installer le module biplist en exécutant la commande suivante :

```
$ pip install biplist
```

4. Ensuite, ouvrez Python et exécutez la commande suivante :

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['CFBundleExecutable']
```

Un package d'application iOS valide doit générer une sortie similaire à la suivante :

```
AWSDeviceFarmiOSReferenceApp
```

Pour de plus amples informations, veuillez consulter [Tests iOS dans AWS Device Farm](#).

XCTest Tests de résolution des problèmes dans AWS Device Farm

La rubrique suivante répertorie les messages d'erreur qui apparaissent lors du téléchargement des XCTest tests et recommande des solutions pour résoudre chaque erreur.

Note

Les instructions ci-dessous présument que vous utilisez MacOS.

XCTEST_TEST_PACKAGE_UNZIP_FAILED

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible d'ouvrir votre fichier ZIP de test. Vérifiez que le fichier est valide et réessayez.

Assurez-vous que vous pouvez décompresser le package de l'application sans erreurs. Dans l'exemple suivant, le nom du package est swiftExampleTests.xctest-1.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Un XCTest package valide doit produire un résultat semblable à ce qui suit :

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

Pour de plus amples informations, veuillez consulter [Intégration de Device Farm à XCTest pour iOS](#).

XCTEST_TEST_PACKAGE_XCTEST_DIR_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver le répertoire .xctest à l'intérieur de votre package. Décompressez votre package de test, vérifiez que le répertoire .xctest s'y trouve, puis réessayez.

Dans l'exemple suivant, le nom du package est swiftExampleTests.xctest-1.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le XCTest package est valide, vous trouverez un répertoire dont le nom est similaire à celui *swiftExampleTests.xctest* du répertoire de travail. Le nom doit se terminer par *.xctest*.

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

Pour de plus amples informations, veuillez consulter [Intégration de Device Farm à XCTest pour iOS](#).

XCTEST_TEST_PACKAGE_PLIST_FILE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver le fichier Info.plist à l'intérieur du répertoire .xctest. Décompressez votre package de test, puis ouvrez le répertoire .xctest, vérifiez que le fichier Info.plist se trouve dans le répertoire, puis réessayez.

Dans l'exemple suivant, le nom du package est swiftExampleTests.xctest-1.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le XCTest package est valide, vous trouverez le *Info.plist* fichier dans le *.xctest* répertoire. Dans notre exemple ci-dessous, le répertoire s'appelle *swiftExampleTests.xctest*.

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

Pour de plus amples informations, veuillez consulter [Intégration de Device Farm à XCTest pour iOS](#).

XCTEST_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver la valeur du nom du package à l'intérieur du fichier Info.plist. Décompressez votre package de test, puis ouvrez le fichier Info.plist, vérifiez que la clé « CFBundle Identifier » est spécifiée et réessayez.

Dans l'exemple suivant, le nom du package est swiftExampleTests.xctest-1.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver le *Info.plist* fichier dans un *.xctest* répertoire comme *swiftExampleTests.xctest* dans notre exemple :

```
.
|-- swiftExampleTests.xctest (directory)
    |-- Info.plist
```

```
`-- (any other files)
```

3. Pour rechercher la valeur du nom du package, vous pouvez ouvrir Info.plist à l'aide de Xcode ou de Python.

Pour Python, vous pouvez installer le module biplist en exécutant la commande suivante :

```
$ pip install biplist
```

4. Ensuite, ouvrez Python et exécutez la commande suivante :

```
import biplist
info_plist = biplist.readPlist('swiftExampleTests.xctest/Info.plist')
print info_plist['CFBundleIdentifier']
```

Un package d'XCtest application valide doit produire un résultat semblable à ce qui suit :

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

Pour de plus amples informations, veuillez consulter [Intégration de Device Farm à XCTest pour iOS](#).

XCTEST_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver la valeur du fichier exécutable à l'intérieur du fichier Info.plist. Décompressez votre package de test, puis ouvrez le fichier Info.plist, vérifiez que la clé « CFBundle Executable » est spécifiée, puis réessayez.

Dans l'exemple suivant, le nom du package est swiftExampleTests.xctest-1.zip.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip swiftExampleTests.xctest-1.zip
```


2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver le *Info.plist* fichier dans un *.xctest* répertoire comme *swiftExampleTests.xctest* dans notre exemple :

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

3. Pour rechercher la valeur du nom du package, vous pouvez ouvrir Info.plist à l'aide de Xcode ou de Python.

Pour Python, vous pouvez installer le module biplist en exécutant la commande suivante :

```
$ pip install biplist
```

4. Ensuite, ouvrez Python et exécutez la commande suivante :

```
import biplist
info_plist = biplist.readPlist('swiftExampleTests.xctest/Info.plist')
print info_plist['CFBundleExecutable']
```

Un package d' XCTest application valide doit produire un résultat semblable à ce qui suit :

```
swiftExampleTests
```

Pour de plus amples informations, veuillez consulter [Intégration de Device Farm à XCTest pour iOS](#).

Résolution des problèmes liés aux tests d' XCTest interface utilisateur dans AWS Device Farm

La rubrique suivante répertorie les messages d'erreur qui apparaissent lors du téléchargement des tests d' XCTest interface utilisateur et recommande des solutions pour résoudre chaque erreur.

 Note

Les instructions ci-dessous sont basées sur Linux x86_64 et Mac.

XCTEST_UI_TEST_PACKAGE_UNZIP_FAILED

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

We could not open your test IPA file. Please verify that the file is valid and try again.

Assurez-vous que vous pouvez décompresser le package de l'application sans erreurs. Dans l'exemple suivant, le nom du package est swift-sample-UI.ipa.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip swift-sample-UI.ipa
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Un package d'application iOS valide doit générer une sortie similaire à la suivante :

```
.
|-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
            |-- `swift-sampleUITests.xctest (directory)
                |-- Info.plist
                |-- `-- (any other files)
            |-- `-- (any other files)
```

Pour de plus amples informations, veuillez consulter [Intégration de l' XCTest interface utilisateur pour iOS à Device Farm](#).

XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

We could not find the Payload directory inside your test package. Please unzip your test package, verify that the Payload directory is inside the package, and try again.

Dans l'exemple suivant, le nom du package est swift-sample-UI.ipa.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip swift-sample-UI.ipa
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le package d' XCTest interface utilisateur est valide, vous trouverez le **Payload** répertoire dans le répertoire de travail.

```
.
|-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
        |   |-- swift-sampleUITests.xctest (directory)
        |       |-- Info.plist
        |       |-- (any other files)
        |-- (any other files)
```

Pour de plus amples informations, veuillez consulter [Intégration de l' XCTest interface utilisateur pour iOS à Device Farm](#).

XCTEST_UI_TEST_PACKAGE_APP_DIR_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

We could not find the .app directory inside the Payload directory. Please unzip your test package and then open the Payload directory, verify that the .app directory is inside the directory, and try again.

Dans l'exemple suivant, le nom du package est swift-sample-UI.ipa.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip swift-sample-UI.ipa
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le package d' XCTest interface utilisateur est valide, vous trouverez un **.app** répertoire comme **swift-sampleUITests-Runner.app** dans notre exemple à l'intérieur du **Payload** répertoire.

```
.
|-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
            |-- swift-sampleUITests.xctest (directory)
                |-- Info.plist
                |-- (any other files)
            |-- (any other files)
```

Pour de plus amples informations, veuillez consulter [Intégration de l' XCTest interface utilisateur pour iOS à Device Farm](#).

XCTEST_UI_TEST_PACKAGE_PLUGINS_DIR_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

We could not find the Plugins directory inside the .app directory. Please unzip your test package and then open the .app directory, verify that the Plugins directory is inside the directory, and try again.

Dans l'exemple suivant, le nom du package est swift-sample-UI.ipa.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip swift-sample-UI.ipa
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le package d' XCTest interface utilisateur est valide, vous trouverez le *Plugins* répertoire dans un *.app* répertoire. Dans notre exemple, le répertoire s'appelle *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)
```

Pour de plus amples informations, veuillez consulter [Intégration de l' XCTest interface utilisateur pour iOS à Device Farm](#).

XCTEST_UI_TEST_PACKAGE_XCTEST_DIR_MISSING_IN_PLUGINS_DIR

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

We could not find the `.xctest` directory inside the plugins directory. Please unzip your test package and then open the plugins directory, verify that the `.xctest` directory is inside the directory, and try again.

Dans l'exemple suivant, le nom du package est `swift-sample-UI.ipa`.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip swift-sample-UI.ipa
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le package d' XCTest interface utilisateur est valide, vous y trouverez un `.xctest Plugins` répertoire. Dans notre exemple, le répertoire s'appelle `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)
```

Pour de plus amples informations, veuillez consulter [Intégration de l' XCTest interface utilisateur pour iOS à Device Farm](#).

XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

We could not find the Info.plist file inside the .app directory. Please unzip your test package and then open the .app directory, verify that the Info.plist file is inside the directory, and try again.

Dans l'exemple suivant, le nom du package est swift-sample-UI.ipa.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip swift-sample-UI.ipa
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le package d' XCTest interface utilisateur est valide, vous trouverez le *Info.plist* fichier dans le *.app* répertoire. Dans notre exemple ci-dessous, le répertoire s'appelle *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)
```

Pour de plus amples informations, veuillez consulter [Intégration de l' XCTest interface utilisateur pour iOS à Device Farm](#).

XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING_IN_XCTEST_DIR

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

We could not find the Info.plist file inside the .xctest directory. Please unzip your test package and then open the .xctest directory, verify that the Info.plist file is inside the directory, and try again.

Dans l'exemple suivant, le nom du package est swift-sample-UI.ipa.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip swift-sample-UI.ipa
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le package d' XCTest interface utilisateur est valide, vous trouverez le *Info.plist* fichier dans le *.xctest* répertoire. Dans notre exemple ci-dessous, le répertoire s'appelle *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)
```

Pour de plus amples informations, veuillez consulter [Intégration de l' XCTest interface utilisateur pour iOS à Device Farm](#).

XCTEST_UI_TEST_PACKAGE_CPU_ARCHITECTURE_VALUE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

We could not the CPU architecture value in the Info.plist file. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the key "UIRequiredDeviceCapabilities" is specified, and try again.

Dans l'exemple suivant, le nom du package est swift-sample-UI.ipa.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip swift-sample-UI.ipa
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver le *Info.plist* fichier dans un *.app* répertoire comme *swift-sampleUITests-Runner.app* dans notre exemple :

```
.
|-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
            |-- swift-sampleUITests.xctest (directory)
                |-- Info.plist
                |-- (any other files)
            |-- (any other files)
```

3. Pour rechercher la valeur de l'architecture d'UC, vous pouvez ouvrir Info.plist à l'aide de Xcode ou de Python.

Pour Python, vous pouvez installer le module biplist en exécutant la commande suivante :

```
$ pip install biplist
```

4. Ensuite, ouvrez Python et exécutez la commande suivante :

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/
Info.plist')
print info_plist['UIRequiredDeviceCapabilities']
```

Un package d'XCtest interface utilisateur valide doit produire un résultat comme celui-ci :

```
['armv7']
```

Pour de plus amples informations, veuillez consulter [Intégration de l' XCTest interface utilisateur pour iOS à Device Farm](#).

XCTEST_UI_TEST_PACKAGE_PLATFORM_VALUE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

We could not find the platform value in the Info.plist. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the key "CFBundleSupportedPlatforms" is specified, and try again.

Dans l'exemple suivant, le nom du package est swift-sample-UI.ipa.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip swift-sample-UI.ipa
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver le *Info.plist* fichier dans un *.app* répertoire comme *swift-sampleUITests-Runner.app* dans notre exemple :

```
.
|-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
        |   |-- swift-sampleUITests.xctest (directory)
        |       |-- Info.plist
        |       |-- (any other files)
        |-- (any other files)
```

3. Pour rechercher la valeur de la plateforme, vous pouvez ouvrir Info.plist à l'aide de Xcode ou de Python.

Pour Python, vous pouvez installer le module biplist en exécutant la commande suivante :

```
$ pip install biplist
```

4. Ensuite, ouvrez Python et exécutez la commande suivante :

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['CFBundleSupportedPlatforms']
```

Un package d' XCTest interface utilisateur valide doit produire un résultat comme celui-ci :

```
['iPhoneOS']
```

Pour de plus amples informations, veuillez consulter [Intégration de l' XCTest interface utilisateur pour iOS à Device Farm](#).

XCTEST_UI_TEST_PACKAGE_WRONG_PLATFORM_DEVICE_VALUE

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

We found the platform device value was wrong in the Info.plist file. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the value of the key "CFBundleSupportedPlatforms" does not contain the keyword "simulator", and try again.

Dans l'exemple suivant, le nom du package est swift-sample-UI.ipa.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip swift-sample-UI.ipa
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver le *Info.plist* fichier dans un *.app* répertoire comme *swift-sampleUITests-Runner.app* dans notre exemple :

```
.
|-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
        |   |-- swift-sampleUITests.xctest (directory)
        |       |-- Info.plist
        |       |-- (any other files)
        |-- (any other files)
```

3. Pour rechercher la valeur de la plateforme, vous pouvez ouvrir Info.plist à l'aide de Xcode ou de Python.

Pour Python, vous pouvez installer le module biplist en exécutant la commande suivante :

```
$ pip install biplist
```

4. Ensuite, ouvrez Python et exécutez la commande suivante :

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['CFBundleSupportedPlatforms']
```

Un package d' XCTest interface utilisateur valide doit produire un résultat comme celui-ci :

```
['iPhoneOS']
```

Si le package d' XCTest interface utilisateur est valide, la valeur ne doit pas contenir le mot `clésimulator`.

Pour de plus amples informations, veuillez consulter [Intégration de l' XCTest interface utilisateur pour iOS à Device Farm](#).

XCTEST_UI_TEST_PACKAGE_FORM_FACTOR_VALUE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

We could not the form factor value in the Info.plist. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the key "UIDeviceFamily" is specified, and try again.

Dans l'exemple suivant, le nom du package est swift-sample-UI.ipa.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip swift-sample-UI.ipa
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver le *Info.plist* fichier dans un *.app* répertoire comme *swift-sampleUITests-Runner.app* dans notre exemple :

```
.
|-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
            |-- swift-sampleUITests.xctest (directory)
                |-- Info.plist
                |-- (any other files)
            |-- (any other files)
```

3. Pour rechercher la valeur du facteur de formulaire, vous pouvez ouvrir Info.plist à l'aide de Xcode ou de Python.

Pour Python, vous pouvez installer le module biplist en exécutant la commande suivante :

```
$ pip install biplist
```

4. Ensuite, ouvrez Python et exécutez la commande suivante :

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
```

```
print info_plist['UIDeviceFamily']
```

Un package d' XCTest interface utilisateur valide doit produire un résultat comme celui-ci :

```
[1, 2]
```

Pour de plus amples informations, veuillez consulter [Intégration de l' XCTest interface utilisateur pour iOS à Device Farm](#).

XCTEST_UI_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

We could not find the package name value in the Info.plist file. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the key "CFBundleIdentifier" is specified, and try again.

Dans l'exemple suivant, le nom du package est swift-sample-UI.ipa.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip swift-sample-UI.ipa
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver le *Info.plist* fichier dans un *.app* répertoire comme *swift-sampleUITests-Runner.app* dans notre exemple :

```
.
|-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
        |
        |-- swift-sampleUITests.xctest (directory)
```

```
|                                |-- Info.plist  
|                                '-- (any other files)  
`-- (any other files)
```

3. Pour rechercher la valeur du nom du package, vous pouvez ouvrir Info.plist à l'aide de Xcode ou de Python.

Pour Python, vous pouvez installer le module biplist en exécutant la commande suivante :

```
$ pip install biplist
```

4. Ensuite, ouvrez Python et exécutez la commande suivante :

```
import biplist  
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')  
print info_plist['CFBundleIdentifier']
```

Un package d' XCTest interface utilisateur valide doit produire un résultat comme celui-ci :

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

Pour de plus amples informations, veuillez consulter [Intégration de l' XCTest interface utilisateur pour iOS à Device Farm](#).

XCTEST_UI_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

We could not find the executable value in the Info.plist file. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the key "CFBundleExecutable" is specified, and try again.

Dans l'exemple suivant, le nom du package est swift-sample-UI.ipa.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip swift-sample-UI.ipa
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver le *Info.plist* fichier dans un *.app* répertoire comme *swift-sampleUITests-Runner.app* dans notre exemple :

```
.
|-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
            |-- swift-sampleUITests.xctest (directory)
                |-- Info.plist
                |-- (any other files)
            |-- (any other files)
```

3. Pour rechercher la valeur du fichier exécutable, vous pouvez ouvrir Info.plist à l'aide de Xcode ou de Python.

Pour Python, vous pouvez installer le module biplist en exécutant la commande suivante :

```
$ pip install biplist
```

4. Ensuite, ouvrez Python et exécutez la commande suivante :

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['CFBundleExecutable']
```

Un package d'XCtest interface utilisateur valide doit produire un résultat comme celui-ci :

```
XCTRunner
```

Pour de plus amples informations, veuillez consulter [Intégration de l'XCtest interface utilisateur pour iOS à Device Farm](#).

XCTEST_UI_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

We could not find the package name value in the Info.plist file inside the .xctest directory. Please unzip your test package and then open the Info.plist file inside the .xctest directory, verify that the key "CFBundleIdentifier" is specified, and try again.

Dans l'exemple suivant, le nom du package est swift-sample-UI.ipa.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip swift-sample-UI.ipa
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver le *Info.plist* fichier dans un *.app* répertoire comme *swift-sampleUITests-Runner.app* dans notre exemple :

```
.
|-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
        |   |-- swift-sampleUITests.xctest (directory)
        |       |-- Info.plist
        |       |-- (any other files)
        |-- (any other files)
```

3. Pour rechercher la valeur du nom du package, vous pouvez ouvrir Info.plist à l'aide de Xcode ou de Python.

Pour Python, vous pouvez installer le module biplist en exécutant la commande suivante :

```
$ pip install biplist
```

4. Ensuite, ouvrez Python et exécutez la commande suivante :

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Plugins/
swift-sampleUITests.xctest/Info.plist')
print info_plist['CFBundleIdentifier']
```

Un package d' XCTest interface utilisateur valide doit produire un résultat comme celui-ci :

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

Pour de plus amples informations, veuillez consulter [Intégration de l' XCTest interface utilisateur pour iOS à Device Farm](#).

XCTEST_UI_TEST_PACKAGE_TEST_EXECUTABLE_VALUE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

We could not find the executable value in the Info.plist file inside the .xctest directory. Please unzip your test package and then open the Info.plist file inside the .xctest directory, verify that the key "CFBundleExecutable" is specified, and try again.

Dans l'exemple suivant, le nom du package est swift-sample-UI.ipa.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip swift-sample-UI.ipa
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Vous devriez trouver le *Info.plist* fichier dans un *.app* répertoire comme *swift-sampleUITests-Runner.app* dans notre exemple :

```
.
```

```

`-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
        |   |-- swift-sampleUITests.xctest (directory)
        |       |-- Info.plist
        |       |-- (any other files)
    |-- (any other files)

```

3. Pour rechercher la valeur du fichier exécutable, vous pouvez ouvrir Info.plist à l'aide de Xcode ou de Python.

Pour Python, vous pouvez installer le module biplist en exécutant la commande suivante :

```
$ pip install biplist
```

4. Ensuite, ouvrez Python et exécutez la commande suivante :

```

import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Plugins/
swift-sampleUITests.xctest/Info.plist')
print info_plist['CFBundleExecutable']

```

Un package d' XCTest interface utilisateur valide doit produire un résultat comme celui-ci :

```
swift-sampleUITests
```

Pour de plus amples informations, veuillez consulter [Intégration de l' XCTest interface utilisateur pour iOS à Device Farm](#).

XCTEST_UI_TEST_PACKAGE_MULTIPLE_APP_DIRS

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

We found multiple .app directories inside your test package. Please unzip your test package, verify that only a single .app directory is present inside the package, then try again.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip swift-sample-UI.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le package d' XCTest interface utilisateur est valide, vous ne devriez trouver qu'un seul .app répertoire, comme `swift-sampleUITests-Runner.app` dans notre exemple, dans le package de test .zip.

```
.
|--swift-sample-UI.zip--(directory)
  |-- swift-sampleUITests-Runner.app (directory)
    |-- Info.plist
    |-- Plugins (directory)
    |   |--swift-sampleUITests.xctest (directory)
    |       |-- Info.plist
    |       |-- (any other files)
    |-- (any other files)
  |-- (any other files)
```

Pour de plus amples informations, veuillez consulter [Intégration de l' XCTest interface utilisateur pour iOS à Device Farm](#).

XCTEST_UI_TEST_PACKAGE_MULTIPLE_IPA_DIRS

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

We found multiple .ipa directories inside your test package. Please unzip your test package, verify that only a single .ipa directory is present inside the package, then try again.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip swift-sample-UI.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le package d' XCTest interface utilisateur est valide, vous ne devriez trouver qu'un seul .ipa répertoire, comme `sampleUITests.ipa` dans notre exemple, dans le package de test .zip.

```
.
|--swift-sample-UI.zip--(directory)
  |-- sampleUITests.ipa (directory)
    |-- Payload (directory)
      |-- swift-sampleUITests-Runner.app (directory)
    |-- (any other files)
```

Pour de plus amples informations, veuillez consulter [Intégration de l' XCTest interface utilisateur pour iOS à Device Farm](#).

XCTEST_UI_TEST_PACKAGE_BOTH_APP_ET_IPA_DIR_PRESENT

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

We found both .app and .ipa files inside your test package. Please unzip your test package, verify that only a single .app or .ipa file is present inside the package, then try again.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip swift-sample-UI.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le package d' XCTest interface utilisateur est valide, vous devriez trouver un .ipa répertoire similaire `sampleUITests.ipa` ou un .app répertoire comme `swift-sampleUITests-`

Runner.app dans notre exemple dans le package de test .zip. Vous pouvez vous référer à un exemple de package de test XCTest_UI valide dans notre documentation sur. [Intégration de l' XCTest interface utilisateur pour iOS à Device Farm](#)

```
.
`--swift-sample-UI.zip--(directory)
    |-- sampleUITests.ipa (directory)
        |-- Payload (directory)
            |-- swift-sampleUITests-Runner.app (directory)
        |-- (any other files)
```

or

```
.
`--swift-sample-UI.zip--(directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
        |-- (any other files)
    |-- (any other files)
```

Pour de plus amples informations, veuillez consulter [Intégration de l' XCTest interface utilisateur pour iOS à Device Farm](#).

XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_PRESENT_DANS_ZIP

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

We found a Payload directory inside your .zip test package. Please unzip your test package, ensure that a Payload directory is not present in the package, then try again.

1. Copiez votre package de test dans votre répertoire de travail, puis exécutez la commande suivante :

```
$ unzip swift-sample-UI.zip
```

2. Pour trouver la structure arborescente du répertoire de travail une fois le package décompressé, exécutez la commande suivante :

```
$ tree .
```

Si le package d' XCTest interface utilisateur est valide, vous ne devriez pas trouver de répertoire de charge utile dans votre package de test.

```
.
|--swift-sample-UI.zip--(directory)
  |-- swift-sampleUITests-Runner.app (directory)
    |-- Info.plist
    |-- Plugins (directory)
    |-- (any other files)
  |-- Payload (directory) [This directory should not be present]
    |-- (any other files)
  |-- (any other files)
```

Pour de plus amples informations, veuillez consulter [Intégration de l' XCTest interface utilisateur pour iOS à Device Farm](#).

Sécurité dans AWS Device Farm

La sécurité du cloud AWS est la priorité absolue. En tant que AWS client, vous bénéficiez d'un centre de données et d'une architecture réseau conçus pour répondre aux exigences des entreprises les plus sensibles en matière de sécurité.

La sécurité est une responsabilité partagée entre vous AWS et vous. Le [modèle de responsabilité partagée](#) décrit ceci comme la sécurité du cloud et la sécurité dans le cloud :

- Sécurité du cloud : AWS est chargée de protéger l'infrastructure qui exécute les AWS services dans le AWS cloud. AWS vous fournit également des services que vous pouvez utiliser en toute sécurité. Des auditeurs tiers testent et vérifient régulièrement l'efficacité de notre sécurité dans le cadre des programmes de [AWS conformité Programmes](#) de de conformité. Pour en savoir plus sur les programmes de conformité applicables à AWS Device Farm, consultez la section [Services AWS concernés par programme de conformité](#) .
- Sécurité dans le cloud : votre responsabilité est déterminée par le service AWS que vous utilisez. Vous êtes également responsable d'autres facteurs, y compris de la sensibilité de vos données, des exigences de votre entreprise, ainsi que de la législation et de la réglementation applicables.

Cette documentation vous aide à comprendre comment appliquer le modèle de responsabilité partagée lors de l'utilisation de Device Farm. Les rubriques suivantes expliquent comment configurer Device Farm pour répondre à vos objectifs de sécurité et de conformité. Vous apprendrez également à utiliser d'autres services AWS qui vous aident à surveiller et à sécuriser les ressources de votre Device Farm.

Rubriques

- [Gestion des identités et des accès dans AWS Device Farm](#)
- [Validation de conformité pour AWS Device Farm](#)
- [Protection des données dans AWS Device Farm](#)
- [Résilience dans AWS Device Farm](#)
- [Sécurité de l'infrastructure dans AWS Device Farm](#)
- [Analyse et gestion des vulnérabilités de configuration dans Device Farm](#)
- [Réponse aux incidents dans Device Farm](#)
- [Journalisation et surveillance dans Device Farm](#)
- [Bonnes pratiques en matière de sécurité pour Device Farm](#)

Gestion des identités et des accès dans AWS Device Farm

Public ciblé

La façon dont vous utilisez Gestion des identités et des accès AWS (IAM) varie en fonction de votre rôle :

- Utilisateur du service : demandez des autorisations à votre administrateur si vous ne pouvez pas accéder aux fonctionnalités (voir [Résolution des problèmes d'identité et d'accès à AWS Device Farm](#))
- Administrateur du service : déterminez l'accès des utilisateurs et soumettez les demandes d'autorisation (voir [Comment AWS Device Farm fonctionne avec IAM](#))
- Administrateur IAM : rédigez des politiques pour gérer l'accès (voir [Exemples de politiques basées sur l'identité d'AWS Device Farm](#))

Authentification par des identités

L'authentification est la façon dont vous vous connectez à AWS l'aide de vos informations d'identification. Vous devez être authentifié en tant qu'utilisateur IAM ou en assumant un rôle IAM. Utilisateur racine d'un compte AWS

Vous pouvez vous connecter en tant qu'identité fédérée à l'aide d'informations d'identification provenant d'une source d'identité telle que AWS IAM Identity Center (IAM Identity Center), d'une authentification unique ou d'informations d'identification. Google/Facebook Pour plus d'informations sur la connexion, consultez [Connexion à votre Compte AWS](#) dans le Guide de l'utilisateur Connexion à AWS .

Pour l'accès par programmation, AWS fournit un SDK et une CLI pour signer les demandes de manière cryptographique. Pour plus d'informations, consultez [Signature AWS Version 4 pour les demandes d'API](#) dans le Guide de l'utilisateur IAM.

Compte AWS utilisateur root

Lorsque vous créez un Compte AWS, vous commencez par une seule identité de connexion appelée utilisateur Compte AWS root qui dispose d'un accès complet à toutes Services AWS les ressources. Il est vivement déconseillé d'utiliser l'utilisateur racine pour vos tâches quotidiennes. Pour les tâches qui requièrent des informations d'identification de l'utilisateur racine, consultez [Tâches qui requièrent les informations d'identification de l'utilisateur racine](#) dans le Guide de l'utilisateur IAM.

Utilisateurs et groupes IAM

Un [utilisateur IAM](#) est une identité qui dispose d'autorisations spécifiques pour une seule personne ou application. Nous vous recommandons d'utiliser ces informations d'identification temporaires au lieu des utilisateurs IAM avec des informations d'identification à long terme. Pour plus d'informations, voir [Exiger des utilisateurs humains qu'ils utilisent la fédération avec un fournisseur d'identité pour accéder à AWS l'aide d'informations d'identification temporaires](#) dans le guide de l'utilisateur IAM.

[Les groupes IAM](#) spécifient une collection d'utilisateurs IAM et permettent de gérer plus facilement les autorisations pour de grands ensembles d'utilisateurs. Pour plus d'informations, consultez [Cas d'utilisation pour les utilisateurs IAM](#) dans le Guide de l'utilisateur IAM.

Rôles IAM

Un [rôle IAM](#) est une identité dotée d'autorisations spécifiques qui fournit des informations d'identification temporaires. Vous pouvez assumer un rôle en [passant d'un rôle d'utilisateur à un rôle IAM \(console\)](#) ou en appelant une opération d' AWS API AWS CLI ou d'API. Pour plus d'informations, consultez [Méthodes pour endosser un rôle](#) dans le Guide de l'utilisateur IAM.

Les rôles IAM sont utiles pour l'accès des utilisateurs fédérés, les autorisations temporaires des utilisateurs IAM, les accès intercompte, les accès entre services et les applications exécutées sur Amazon EC2. Pour plus d'informations, consultez [Accès intercompte aux ressources dans IAM](#) dans le Guide de l'utilisateur IAM.

Comment AWS Device Farm fonctionne avec IAM

Avant d'utiliser IAM pour gérer l'accès à Device Farm, vous devez savoir quelles fonctionnalités IAM peuvent être utilisées avec Device Farm. Pour obtenir une vue d'ensemble de la manière dont Device Farm et les autres AWS services fonctionnent avec IAM, consultez la section [AWS Services That Work with IAM](#) dans le guide de l'utilisateur IAM.

Rubriques

- [Politiques basées sur l'identité de Device Farm](#)
- [Politiques basées sur les ressources de Device Farm](#)
- [Listes de contrôle d'accès \(ACL\)](#)
- [Autorisation basée sur les tags Device Farm](#)
- [Rôles IAM de Device Farm](#)

Politiques basées sur l'identité de Device Farm

Avec les stratégies IAM basées sur l'identité, vous pouvez spécifier des actions et ressources autorisées ou refusées et les conditions dans lesquelles les actions sont autorisées ou refusées. Device Farm prend en charge des actions, des ressources et des clés de condition spécifiques. Pour en savoir plus sur tous les éléments que vous utilisez dans une politique JSON, consultez [Références des éléments de politique JSON IAM](#) dans le Guide de l'utilisateur IAM.

Actions

Les administrateurs peuvent utiliser les politiques AWS JSON pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément `Action` d'une politique JSON décrit les actions que vous pouvez utiliser pour autoriser ou refuser l'accès à une politique. Intégration d'actions dans une politique afin d'accorder l'autorisation d'exécuter les opérations associées.

Les actions de politique dans Device Farm utilisent le préfixe suivant avant l'action :`devicefarm:`. Par exemple, pour autoriser quelqu'un à démarrer des sessions Selenium avec le fonctionnement de l'`CreateTestGridUrlAPI` de test du navigateur de bureau Device Farm, vous devez inclure l'`devicefarm:CreateTestGridUrl`action dans la politique. Les déclarations de politique doivent inclure un élément `Action` ou `NotAction`. Device Farm définit son propre ensemble d'actions décrivant les tâches que vous pouvez effectuer avec ce service.

Pour spécifier plusieurs actions dans une seule déclaration, séparez-les par des virgules comme suit :

```
"Action": [  
    "devicefarm:action1",  
    "devicefarm:action2"
```

Vous pouvez aussi spécifier plusieurs actions à l'aide de caractères génériques (*). Par exemple, pour spécifier toutes les actions qui commencent par le mot `List`, incluez l'action suivante :

```
"Action": "devicefarm:List*"
```

Pour consulter la liste des actions Device Farm, reportez-vous à la section [Actions définies par AWS Device Farm](#) dans le IAM Service Authorization Reference.

Ressources

Les administrateurs peuvent utiliser les politiques AWS JSON pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément de politique JSON `Resource` indique le ou les objets auxquels l'action s'applique. Il est recommandé de définir une ressource à l'aide de son [Amazon Resource Name \(ARN\)](#). Pour les actions qui ne sont pas compatibles avec les autorisations de niveau ressource, utilisez un caractère générique (*) afin d'indiquer que l'instruction s'applique à toutes les ressources.

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

La ressource d'instance Amazon EC2 possède l'ARN suivant :

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

Pour plus d'informations sur le format de ARNs, consultez [Amazon Resource Names \(ARNs\) et AWS Service Namespaces](#).

Par exemple, pour spécifier l'instance `i-1234567890abcdef0` dans votre instruction, utilisez l'ARN suivant :

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:instance/i-1234567890abcdef0"
```

Pour spécifier toutes les instances qui appartiennent à un compte, utilisez le caractère générique (*) :

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:instance/*"
```

Certaines actions de Device Farm, telles que celles relatives à la création de ressources, ne peuvent pas être effectuées sur une ressource. Dans ce cas, vous devez utiliser le caractère générique (*).

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

De nombreuses actions d'API Amazon EC2 nécessitent plusieurs ressources. Par exemple, comme `AttachVolume` attache un volume Amazon EBS à une instance, un utilisateur IAM doit avoir les autorisations nécessaires pour utiliser le volume et l'instance. Pour spécifier plusieurs ressources dans une seule instruction, séparez-les ARNs par des virgules.

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

Pour consulter la liste des types de ressources Device Farm et leurs caractéristiques ARNs, reportez-vous à la section [Types de ressources définis par AWS Device Farm](#) dans le manuel IAM Service Authorization Reference. Pour savoir avec quelles actions vous pouvez spécifier l'ARN de chaque ressource, consultez la section [Actions définies par AWS Device Farm](#) dans la référence d'autorisation du service IAM.

Clés de condition

Les administrateurs peuvent utiliser les politiques AWS JSON pour spécifier qui a accès à quoi. C'est-à-dire, quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

L'élément `Condition` indique à quel moment les instructions s'exécutent en fonction de critères définis. Vous pouvez créer des expressions conditionnelles qui utilisent des [opérateurs de condition](#), tels que les signes égal ou inférieur à, pour faire correspondre la condition de la politique aux valeurs de la demande. Pour voir toutes les clés de condition AWS globales, voir les clés de [contexte de condition AWS globales](#) dans le guide de l'utilisateur IAM.

Device Farm définit son propre ensemble de clés de condition et prend également en charge l'utilisation de certaines clés de condition globales. Pour voir toutes les clés de condition AWS globales, consultez la section [Clés contextuelles de condition AWS globale](#) dans le guide de l'utilisateur IAM.

Pour consulter la liste des clés de condition de Device Farm, reportez-vous à la section [Clés de condition AWS Device Farm](#) dans la référence d'autorisation du service IAM. Pour savoir avec quelles actions et ressources vous pouvez utiliser une clé de condition, consultez la section [Actions définies par AWS Device Farm](#) dans la référence d'autorisation du service IAM.

Exemples

Pour consulter des exemples de politiques basées sur l'identité de Device Farm, consultez. [Exemples de politiques basées sur l'identité d'AWS Device Farm](#)

Politiques basées sur les ressources de Device Farm

Device Farm ne prend pas en charge les politiques basées sur les ressources.

Listes de contrôle d'accès (ACL)

Device Farm ne prend pas en charge les listes de contrôle d'accès (ACLs).

Autorisation basée sur les tags Device Farm

Vous pouvez associer des tags aux ressources de Device Farm ou transmettre des tags dans une demande à Device Farm. Pour contrôler l'accès basé sur des étiquettes, vous devez fournir les informations d'étiquette dans l'[élément de condition](#) d'une politique utilisant les clés de condition `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` ou `aws:TagKeys`. Pour plus d'informations sur le balisage des ressources Device Farm, consultez [Marquage dans Device Farm](#).

Pour visualiser un exemple de politique basée sur l'identité permettant de limiter l'accès à une ressource en fonction des balises de cette ressource, consultez [Visualisation des projets de test du navigateur de bureau Device Farm basés sur des balises](#).

Rôles IAM de Device Farm

Un [rôle IAM](#) est une entité de votre AWS compte dotée d'autorisations spécifiques.

Utilisation d'informations d'identification temporaires avec Device Farm

Device Farm prend en charge l'utilisation d'informations d'identification temporaires.

Vous pouvez utiliser des informations d'identification temporaires pour vous connecter à la fédération afin d'assumer un rôle IAM ou un rôle entre comptes. Vous obtenez des informations d'identification de sécurité temporaires en appelant des opérations d' AWS STS API telles que [AssumeRole](#) ou [GetFederationToken](#).

Rôles liés à un service

Les [rôles liés aux](#) AWS services permettent aux services d'accéder aux ressources d'autres services pour effectuer une action en votre nom. Les rôles liés à un service s'affichent dans votre compte IAM et sont la propriété du service. Un administrateur IAM peut consulter, mais ne peut pas modifier, les autorisations pour les rôles liés à un service.

Device Farm utilise des rôles liés aux services dans la fonctionnalité de test du navigateur de bureau Device Farm. Pour plus d'informations sur ces rôles, consultez la section [Using Service-Linked Roles in Device Farm desktop browser testing](#) dans le guide du développeur.

Rôles du service

Device Farm ne prend pas en charge les rôles de service.

Cette fonctionnalité permet à un service d'endosser un [rôle de service](#) en votre nom. Ce rôle autorise le service à accéder à des ressources d'autres services pour effectuer une action en votre nom. Les rôles de service s'affichent dans votre compte IAM et sont la propriété du compte. Cela signifie qu'un administrateur IAM peut modifier les autorisations associées à ce rôle. Toutefois, une telle action peut perturber le bon fonctionnement du service.

Gestion de l'accès à l'aide de politiques

Vous contrôlez l'accès en AWS créant des politiques et en les associant à AWS des identités ou à des ressources. Une politique définit les autorisations lorsqu'elles sont associées à une identité ou à une ressource. AWS évalue ces politiques lorsqu'un directeur fait une demande. La plupart des politiques sont stockées AWS sous forme de documents JSON. Pour plus d'informations les documents de politique JSON, consultez [Vue d'ensemble des politiques JSON](#) dans le Guide de l'utilisateur IAM.

À l'aide de politiques, les administrateurs précisent qui a accès à quoi en définissant quel principal peut effectuer des actions sur quelles ressources et dans quelles conditions.

Par défaut, les utilisateurs et les rôles ne disposent d'aucune autorisation. Un administrateur IAM crée des politiques IAM et les ajoute aux rôles, que les utilisateurs peuvent ensuite assumer. Les politiques IAM définissent les autorisations quelle que soit la méthode que vous utilisez pour exécuter l'opération.

Politiques basées sur l'identité

Les stratégies basées sur l'identité sont des documents de stratégie d'autorisations JSON que vous attachez à une identité (utilisateur, groupe ou rôle). Ces politiques contrôlent les actions que peuvent exécuter ces identités, sur quelles ressources et dans quelles conditions. Pour découvrir comment créer une politique basée sur l'identité, consultez [Définition d'autorisations IAM personnalisées avec des politiques gérées par le client](#) dans le Guide de l'utilisateur IAM.

Les politiques basées sur l'identité peuvent être des politiques intégrées (intégrées directement dans une seule identité) ou des politiques gérées (politiques autonomes associées à plusieurs identités). Pour découvrir comment choisir entre des politiques gérées et en ligne, consultez [Choix entre les politiques gérées et les politiques en ligne](#) dans le Guide de l'utilisateur IAM.

Le tableau suivant décrit les politiques gérées par Device Farm AWS.

Modifier	Description	Date
AWSDeviceFarmFullAccess	Fournit un accès complet à toutes les opérations d'AWS Device Farm.	15 juillet 2015
AWSServiceRoleForDeviceFarmTestGrid	Permet à Device Farm d'accéder aux ressources AWS en votre nom.	20 mai 2021

Autres types de politique

AWS prend en charge des types de politiques supplémentaires qui peuvent définir les autorisations maximales accordées par les types de politiques les plus courants :

- Limites d'autorisations : une limite des autorisations définit le nombre maximum d'autorisations qu'une politique basée sur l'identité peut accorder à une entité IAM. Pour plus d'informations, consultez [Limites d'autorisations pour des entités IAM](#) dans le Guide de l'utilisateur IAM.
- Politiques de contrôle des services (SCPs) — Spécifiez les autorisations maximales pour une organisation ou une unité organisationnelle dans AWS Organizations. Pour plus d'informations, consultez [Politiques de contrôle de service](#) dans le Guide de l'utilisateur AWS Organizations .
- Politiques de contrôle des ressources (RCPs) : définissez le maximum d'autorisations disponibles pour les ressources de vos comptes. Pour plus d'informations, voir [Politiques de contrôle des ressources \(RCPs\)](#) dans le guide de AWS Organizations l'utilisateur.
- Politiques de session : politiques avancées que vous passez en tant que paramètre lorsque vous créez par programmation une session temporaire pour un rôle ou un utilisateur fédéré. Pour plus d'informations, consultez [Politiques de session](#) dans le Guide de l'utilisateur IAM.

Plusieurs types de politique

Lorsque plusieurs types de politiques s'appliquent à la requête, les autorisations en résultant sont plus compliquées à comprendre. Pour savoir comment AWS déterminer s'il faut autoriser une demande lorsque plusieurs types de politiques sont impliqués, consultez la section [Logique d'évaluation des politiques](#) dans le guide de l'utilisateur IAM.

Exemples de politiques basées sur l'identité d'AWS Device Farm

Par défaut, les utilisateurs et les rôles IAM ne sont pas autorisés à créer ou à modifier les ressources Device Farm. Ils ne peuvent pas non plus effectuer de tâches à l'aide de l' AWS API AWS Management Console AWS CLI, ou. Un administrateur IAM doit créer des politiques IAM autorisant les utilisateurs et les rôles à exécuter des opérations d'API spécifiques sur les ressources spécifiées dont ils ont besoin. Il doit ensuite attacher ces politiques aux utilisateurs ou aux groupes IAM ayant besoin de ces autorisations.

Pour savoir comment créer une stratégie IAM basée sur l'identité à l'aide de ces exemples de documents de stratégie JSON, veuillez consulter [Création de stratégies dans l'onglet JSON](#) dans le Guide de l'utilisateur IAM.

Rubriques

- [Bonnes pratiques en matière de politiques](#)
- [Autorisation accordée aux utilisateurs pour afficher leurs propres autorisations](#)
- [Accès à un projet de test de navigateur de bureau Device Farm](#)
- [Visualisation des projets de test du navigateur de bureau Device Farm basés sur des balises](#)

Bonnes pratiques en matière de politiques

Les politiques basées sur l'identité déterminent si quelqu'un peut créer, accéder ou supprimer les ressources Device Farm de votre compte. Ces actions peuvent entraîner des frais pour votre Compte AWS. Lorsque vous créez ou modifiez des politiques basées sur l'identité, suivez ces instructions et recommandations :

- Commencez AWS par les politiques gérées et passez aux autorisations du moindre privilège : pour commencer à accorder des autorisations à vos utilisateurs et à vos charges de travail, utilisez les politiques AWS gérées qui accordent des autorisations pour de nombreux cas d'utilisation courants. Ils sont disponibles dans votre Compte AWS. Nous vous recommandons de réduire davantage les autorisations en définissant des politiques gérées par les AWS clients spécifiques à vos cas d'utilisation. Pour plus d'informations, consultez [politiques gérées par AWS](#) ou [politiques gérées par AWS pour les activités professionnelles](#) dans le Guide de l'utilisateur IAM.
- Accordez les autorisations de moindre privilège : lorsque vous définissez des autorisations avec des politiques IAM, accordez uniquement les autorisations nécessaires à l'exécution d'une seule tâche. Pour ce faire, vous définissez les actions qui peuvent être entreprises sur des ressources spécifiques dans des conditions spécifiques, également appelées autorisations de moindre

privilège. Pour plus d'informations sur l'utilisation d'IAM pour appliquer des autorisations, consultez [politiques et autorisations dans IAM](#) dans le Guide de l'utilisateur IAM.

- Utilisez des conditions dans les politiques IAM pour restreindre davantage l'accès : vous pouvez ajouter une condition à vos politiques afin de limiter l'accès aux actions et aux ressources. Par exemple, vous pouvez écrire une condition de politique pour spécifier que toutes les demandes doivent être envoyées via SSL. Vous pouvez également utiliser des conditions pour accorder l'accès aux actions de service si elles sont utilisées par le biais d'un service spécifique Service AWS, tel que CloudFormation. Pour plus d'informations, consultez [Conditions pour éléments de politique JSON IAM](#) dans le Guide de l'utilisateur IAM.
- Utilisez l'Analyseur d'accès IAM pour valider vos politiques IAM afin de garantir des autorisations sécurisées et fonctionnelles : l'Analyseur d'accès IAM valide les politiques nouvelles et existantes de manière à ce que les politiques IAM respectent le langage de politique IAM (JSON) et les bonnes pratiques IAM. IAM Access Analyzer fournit plus de 100 vérifications de politiques et des recommandations exploitables pour vous aider à créer des politiques sécurisées et fonctionnelles. Pour plus d'informations, consultez [Validation de politiques avec IAM Access Analyzer](#) dans le Guide de l'utilisateur IAM.
- Exiger l'authentification multifactorielle (MFA) : si vous avez un scénario qui nécessite des utilisateurs IAM ou un utilisateur root, activez l'authentification MFA pour une sécurité accrue. Compte AWS Pour exiger la MFA lorsque des opérations d'API sont appelées, ajoutez des conditions MFA à vos politiques. Pour plus d'informations, consultez [Sécurisation de l'accès aux API avec MFA](#) dans le Guide de l'utilisateur IAM.

Pour plus d'informations sur les bonnes pratiques dans IAM, consultez [Bonnes pratiques de sécurité dans IAM](#) dans le Guide de l'utilisateur IAM.

Autorisation accordée aux utilisateurs pour afficher leurs propres autorisations

Cet exemple montre comment créer une politique qui permet aux utilisateurs IAM d'afficher les politiques en ligne et gérées attachées à leur identité d'utilisateur. Cette politique inclut les autorisations permettant d'effectuer cette action sur la console ou par programmation à l'aide de l'API AWS CLI or AWS .

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
```

```

        "Effect": "Allow",
        "Action": [
            "iam:GetUserPolicy",
            "iam:ListGroupsForUser",
            "iam:ListAttachedUserPolicies",
            "iam:ListUserPolicies",
            "iam:GetUser"
        ],
        "Resource": ["arn:aws:iam::*:user/${aws:username}"]
    },
    {
        "Sid": "NavigateInConsole",
        "Effect": "Allow",
        "Action": [
            "iam:GetGroupPolicy",
            "iam:GetPolicyVersion",
            "iam:GetPolicy",
            "iam:ListAttachedGroupPolicies",
            "iam:ListGroupPolicies",
            "iam:ListPolicyVersions",
            "iam:ListPolicies",
            "iam:ListUsers"
        ],
        "Resource": "*"
    }
]
}

```

Accès à un projet de test de navigateur de bureau Device Farm

Dans cet exemple, vous souhaitez autoriser un utilisateur IAM de votre AWS compte à accéder à l'un de vos projets de test du navigateur de bureau Device Farm. `arn:aws:devicefarm:us-west-2:111122223333:testgrid-project:123e4567-e89b-12d3-a456-426655441111` Vous souhaitez que le compte puisse voir les éléments associés au projet.

En plus du point de terminaison `devicefarm:GetTestGridProject`, le compte doit avoir les points de terminaison `devicefarm:ListTestGridSessions`, `devicefarm:GetTestGridSession`, `devicefarm:ListTestGridSessionActions` et `devicefarm:ListTestGridSessionArtifacts`.

Si vous utilisez des systèmes CI, vous devez donner des informations d'identification d'accès uniques à chaque exécuteur CI. Par exemple, il est peu probable qu'un système CI ait besoin d'autorisations

autres que `devicefarm:ScheduleRun` ou `devicefarm:CreateUpload`. La politique IAM suivante décrit une politique minimale permettant à un utilisateur de CI de démarrer le test d'une nouvelle application native Device Farm en créant un téléchargement et en l'utilisant pour planifier un test :

Visualisation des projets de test du navigateur de bureau Device Farm basés sur des balises

Vous pouvez utiliser les conditions de votre politique basée sur l'identité pour contrôler l'accès aux ressources de Device Farm en fonction de balises. Cet exemple montre comment créer une stratégie qui autorise l'affichage de projets et de sessions. L'autorisation est accordée si la balise `Owner` de la ressource demandée correspond au nom d'utilisateur du compte demandeur.

Vous pouvez rattacher cette politique aux utilisateurs IAM de votre compte. Si un utilisateur nommé `richard-roe` tente de consulter un projet ou une session Device Farm, le projet doit être étiqueté `Owner=richard-roe` ou `owner=richard-roe`. Dans le cas contraire, l'utilisateur se voit refuser l'accès. La clé de condition de balise `Owner` correspond à la fois à `Owner` et à `owner`, car les noms de clé de condition ne sont pas sensibles à la casse. Pour plus d'informations, consultez [Éléments de politique JSON IAM : Condition](#) dans le Guide de l'utilisateur IAM.

Résolution des problèmes d'identité et d'accès à AWS Device Farm

Utilisez les informations suivantes pour vous aider à diagnostiquer et à résoudre les problèmes courants que vous pouvez rencontrer lorsque vous travaillez avec Device Farm et IAM.

Je ne suis pas autorisé à effectuer une action dans Device Farm

Si vous recevez un message d'erreur indiquant AWS Management Console que vous n'êtes pas autorisé à effectuer une action, vous devez contacter votre administrateur pour obtenir de l'aide. Votre administrateur est la personne qui vous a fourni votre nom d'utilisateur et votre mot de passe.

L'exemple d'erreur suivant se produit lorsque l'utilisateur IAM essaie d'utiliser la console pour afficher les détails d'une exécution, mais ne dispose pas des `devicefarm:GetRun` autorisations nécessaires. `mateojackson`

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
devicefarm:GetRun on resource: arn:aws:devicefarm:us-west-2:123456789101:run:123e4567-
e89b-12d3-a456-426655440000/123e4567-e89b-12d3-a456-426655441111
```

Dans ce cas, Mateo demande à son administrateur de mettre à jour ses stratégies pour lui permettre d'accéder à la ressource `devicefarm:GetRuns` sur la ressource `arn:aws:devicefarm:us-west-2:123456789101:run:123e4567-e89b-12d3-a456-426655440000/123e4567-e89b-12d3-a456-426655441111` à l'aide de l'action `devicefarm:GetRun`.

Je ne suis pas autorisé à effectuer `iam:PassRole`

Si vous recevez un message d'erreur indiquant que vous n'êtes pas autorisé à effectuer l'`iam:PassRole` action, vos politiques doivent être mises à jour pour vous permettre de transmettre un rôle à Device Farm.

Certains services AWS permettent de transmettre un rôle existant à ce service au lieu de créer un nouveau rôle de service ou un rôle lié à un service. Pour ce faire, vous devez disposer des autorisations nécessaires pour transmettre le rôle au service.

L'exemple d'erreur suivant se produit lorsqu'un utilisateur IAM nommé `marymajor` essaie d'utiliser la console pour effectuer une action dans Device Farm. Toutefois, l'action nécessite que le service ait des autorisations accordées par un rôle de service. Mary n'est pas autorisée à transmettre le rôle au service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

Dans ce cas, les politiques de Mary doivent être mises à jour pour lui permettre d'exécuter l'action `iam:PassRole`.

Si vous avez besoin d'aide, contactez votre AWS administrateur. Votre administrateur vous a fourni vos informations de connexion.

Je veux afficher mes clés d'accès

Une fois les clés d'accès utilisateur IAM créées, vous pouvez afficher votre ID de clé d'accès à tout moment. Toutefois, vous ne pouvez pas revoir votre clé d'accès secrète. Si vous perdez votre clé d'accès secrète, vous devez créer une nouvelle paire de clés.

Les clés d'accès se composent de deux parties : un ID de clé d'accès (par exemple, `AKIAIOSFODNN7EXAMPLE`) et une clé d'accès secrète (par exemple, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). À l'instar d'un nom d'utilisateur et un mot de passe, vous devez utiliser à la fois l'ID de clé d'accès et la clé d'accès secrète pour authentifier vos demandes. Gérez vos clés d'accès de manière aussi sécurisée que votre nom d'utilisateur et votre mot de passe.

 Important

Ne communiquez pas vos clés d'accès à un tiers, même pour qu'il vous aide à [trouver votre ID utilisateur canonique](#). Ce faisant, vous pourriez donner à quelqu'un un accès permanent à votre Compte AWS.

Lorsque vous créez une paire de clé d'accès, enregistrez l'ID de clé d'accès et la clé d'accès secrète dans un emplacement sécurisé. La clé d'accès secrète est accessible uniquement au moment de sa création. Si vous perdez votre clé d'accès secrète, vous devez ajouter de nouvelles clés d'accès pour votre utilisateur IAM. Vous pouvez avoir un maximum de deux clés d'accès. Si vous en avez déjà deux, vous devez supprimer une paire de clés avant d'en créer une nouvelle. Pour afficher les instructions, consultez [Gestion des clés d'accès](#) dans le Guide de l'utilisateur IAM.

Je suis administrateur et je souhaite autoriser d'autres personnes à accéder à Device Farm

Pour autoriser d'autres personnes à accéder à Device Farm, vous devez autoriser les personnes ou les applications qui ont besoin d'y accéder. Si vous utilisez AWS IAM Identity Center pour gérer des personnes et des applications, vous attribuez des ensembles d'autorisations aux utilisateurs ou aux groupes afin de définir leur niveau d'accès. Les ensembles d'autorisations créent et attribuent automatiquement des politiques IAM aux rôles IAM associés à la personne ou à l'application. Pour plus d'informations, consultez la section [Ensembles d'autorisations](#) dans le guide de AWS IAM Identity Center l'utilisateur.

Si vous n'utilisez pas IAM Identity Center, vous devez créer des entités IAM (utilisateurs ou rôles) pour les personnes ou les applications qui ont besoin d'un accès. Vous devez ensuite associer une politique à l'entité qui lui accorde les autorisations appropriées dans Device Farm. Une fois les autorisations accordées, fournissez les informations d'identification à l'utilisateur ou au développeur de l'application. Ils utiliseront ces informations d'identification pour y accéder AWS. Pour en savoir plus sur la création d'utilisateurs, de groupes, de politiques et d'autorisations [IAM, consultez la section Identités, politiques et autorisations IAM dans le guide de l'utilisateur IAM](#).

Je souhaite autoriser des personnes extérieures à mon AWS compte à accéder aux ressources de mon Device Farm

Vous pouvez créer un rôle que les utilisateurs provenant d'autres comptes ou les personnes extérieures à votre organisation pourront utiliser pour accéder à vos ressources. Vous pouvez

spécifier qui est autorisé à assumer le rôle. Pour les services qui prennent en charge les politiques basées sur les ressources ou les listes de contrôle d'accès (ACLs), vous pouvez utiliser ces politiques pour autoriser les utilisateurs à accéder à vos ressources.

Pour plus d'informations, consultez les éléments suivants :

- Pour savoir si Device Farm prend en charge ces fonctionnalités, consultez [Comment AWS Device Farm fonctionne avec IAM](#).
- Pour savoir comment fournir l'accès à vos ressources sur celles Comptes AWS que vous possédez, consultez la section [Fournir l'accès à un utilisateur IAM dans un autre utilisateur Compte AWS que vous possédez](#) dans le Guide de l'utilisateur IAM.
- Pour savoir comment fournir l'accès à vos ressources à des tiers Comptes AWS, consultez la section [Fournir un accès à des ressources Comptes AWS détenues par des tiers](#) dans le guide de l'utilisateur IAM.
- Pour savoir comment fournir un accès par le biais de la fédération d'identité, consultez [Fournir un accès à des utilisateurs authentifiés en externe \(fédération d'identité\)](#) dans le Guide de l'utilisateur IAM.
- Pour en savoir plus sur la différence entre l'utilisation des rôles et des politiques basées sur les ressources pour l'accès intercompte, consultez [Accès intercompte aux ressources dans IAM](#) dans le Guide de l'utilisateur IAM.

Validation de conformité pour AWS Device Farm

Des auditeurs tiers évaluent la sécurité et la conformité dans AWS Device Farm le cadre de plusieurs programmes de AWS conformité. Il s'agit notamment du SOC, du PCI, du FedRAMP, de l'HIPAA et d'autres. AWS Device Farm n'entre dans le champ d'aucun programme de AWS conformité.

Pour obtenir la liste des AWS services concernés par des programmes de conformité spécifiques, voir [Services AWS concernés par programme de conformité](#) . Pour des informations générales, voir Programmes de [AWS conformité Programmes AWS](#) de .

Vous pouvez télécharger des rapports d'audit tiers à l'aide de AWS Artifact. Pour plus d'informations, consultez [Téléchargement de rapports dans AWS Artifact](#).

Lorsque vous utilisez Device Farm, votre responsabilité en matière de conformité dépend de la sensibilité de vos données, des objectifs de conformité de votre entreprise et des lois et réglementations applicables. AWS fournit les ressources suivantes pour faciliter la mise en conformité :

- [Guides démarrage rapide de la sécurité et de la conformité](#). Ces guides de déploiement traitent des considérations architecturales et fournissent des étapes pour déployer des environnements de base axés sur la sécurité et la conformité sur AWS.
- AWS Ressources de <https://aws.amazon.com/compliance/resources/> de conformité — Cette collection de classeurs et de guides peut s'appliquer à votre secteur d'activité et à votre région.
- [Évaluation des ressources à l'aide des règles](#) du guide du AWS Config développeur : AWS Config évalue dans quelle mesure les configurations de vos ressources sont conformes aux pratiques internes, aux directives du secteur et aux réglementations.
- [AWS Security Hub CSPM](#)— Ce AWS service fournit une vue complète de l'état de votre sécurité interne, AWS ce qui vous permet de vérifier votre conformité aux normes et aux meilleures pratiques du secteur de la sécurité.

Protection des données dans AWS Device Farm

Le [modèle de responsabilité AWS partagée](#) s'applique à la protection des données dans AWS Device Farm (Device Farm). Comme décrit dans ce modèle, AWS est chargé de protéger l'infrastructure mondiale qui gère tous les AWS Cloud. La gestion du contrôle de votre contenu hébergé sur cette infrastructure relève de votre responsabilité. Vous êtes également responsable des tâches de configuration et de gestion de la sécurité des Services AWS que vous utilisez. Pour plus d'informations sur la confidentialité des données, consultez [Questions fréquentes \(FAQ\) sur la confidentialité des données](#). Pour en savoir plus sur la protection des données en Europe, consultez le billet de blog [Modèle de responsabilité partagée d'AWS et RGPD \(Règlement général sur la protection des données\)](#) sur le Blog de sécuritéAWS .

À des fins de protection des données, nous vous recommandons de protéger les Compte AWS informations d'identification et de configurer les utilisateurs individuels avec AWS IAM Identity Center ou Gestion des identités et des accès AWS (IAM). Ainsi, chaque utilisateur se voit attribuer uniquement les autorisations nécessaires pour exécuter ses tâches. Nous vous recommandons également de sécuriser vos données comme indiqué ci-dessous :

- Utilisez l'authentification multifactorielle (MFA) avec chaque compte.
- SSL/TLS À utiliser pour communiquer avec AWS les ressources. Nous exigeons TLS 1.2 et recommandons TLS 1.3.
- Configurez l'API et la journalisation de l'activité des utilisateurs avec AWS CloudTrail. Pour plus d'informations sur l'utilisation des CloudTrail sentiers pour capturer AWS des activités, consultez la section [Utilisation des CloudTrail sentiers](#) dans le guide de AWS CloudTrail l'utilisateur.

- Utilisez des solutions de AWS chiffrement, ainsi que tous les contrôles de sécurité par défaut qu'ils contiennent Services AWS.
- Utilisez des services de sécurité gérés avancés tels qu'Amazon Macie, qui contribuent à la découverte et à la sécurisation des données sensibles stockées dans Amazon S3.
- Si vous avez besoin de modules cryptographiques validés par la norme FIPS 140-3 pour accéder AWS via une interface de ligne de commande ou une API, utilisez un point de terminaison FIPS. Pour plus d'informations sur les points de terminaison FIPS disponibles, consultez [Norme FIPS \(Federal Information Processing Standard\) 140-3](#).

Nous vous recommandons fortement de ne jamais placer d'informations confidentielles ou sensibles, telles que les adresses e-mail de vos clients, dans des balises ou des champs de texte libre tels que le champ Nom. Cela inclut lorsque vous travaillez avec Device Farm ou autre Services AWS à l'aide de la console, de l'API ou AWS SDKs. AWS CLI Toutes les données que vous entrez dans des balises ou des champs de texte de forme libre utilisés pour les noms peuvent être utilisées à des fins de facturation ou dans les journaux de diagnostic. Si vous fournissez une adresse URL à un serveur externe, nous vous recommandons fortement de ne pas inclure d'informations d'identification dans l'adresse URL permettant de valider votre demande adressée à ce serveur.

Chiffrement en transit

Les points de terminaison Device Farm ne prennent en charge que le protocole HTTPS signé (SSL/TLS) requests except where otherwise noted. All content retrieved from or placed in Amazon S3 through upload URLs is encrypted using SSL/TLS. Pour plus d'informations sur la façon dont les requêtes HTTPS sont connectées AWS, consultez la section [Signature des demandes AWS d'API](#) dans le manuel de référence AWS général.

Il est de votre responsabilité de chiffrer et de sécuriser toutes les communications effectuées par vos applications testées et par toutes les applications supplémentaires installée dans le cadre de l'exécution des tests sur les périphériques.

Chiffrement au repos

La fonctionnalité de test du navigateur de bureau de Device Farm prend en charge le chiffrement au repos des artefacts générés lors des tests.

Les données de test des appareils mobiles physiques de Device Farm ne sont pas cryptées au repos.

Conservation des données

Les données de Device Farm sont conservées pendant une durée limitée. Une fois la période de conservation expirée, les données sont supprimées du stockage de sauvegarde de Device Farm.

Type de contenu	Période de conservation (jours)	Période de conservation des métadonnées (jours)
Applications chargées	30	30
Paquets de test chargés	30	30
Journaux	400	400
Enregistrements vidéo et autres artefacts	400	400

Il vous incombe d'archiver tout contenu que vous souhaitez conserver pendant des périodes plus longues.

Gestion des données

Les données de Device Farm sont gérées différemment selon les fonctionnalités utilisées. Cette section explique comment les données sont gérées pendant et après l'utilisation de Device Farm.

Test du navigateur de bureau

Les instances utilisées pendant les sessions Selenium ne sont pas enregistrées. Toutes les données générées à la suite d'interactions du navigateur sont supprimées en fin de session.

Cette fonctionnalité prend actuellement en charge le chiffrement au repos pour les artefacts générés pendant le test.

Tests d'appareils physiques

Les sections suivantes fournissent des informations sur les étapes AWS à suivre pour nettoyer ou détruire les appareils après avoir utilisé Device Farm.

Les données de test des appareils mobiles physiques de Device Farm ne sont pas cryptées au repos.

Flottes d'appareils publics

Une fois l'exécution du test terminée, Device Farm exécute une série de tâches de nettoyage sur chaque appareil du parc d'appareils publics, y compris la désinstallation de votre application. Si nous ne pouvons pas vérifier la désinstallation de votre application ou l'une des autres étapes de nettoyage, l'appareil fait l'objet d'une réinitialisation d'usine avant d'être remis en utilisation.

Note

Il est possible que les données persistent entre les sessions dans certains cas, en particulier si vous utilisez le système de l'appareil en dehors du contexte de votre application. Pour cette raison, et dans la mesure où Device Farm enregistre des vidéos et des journaux d'activité pendant que vous utilisez chaque appareil, nous vous recommandons de ne pas saisir d'informations sensibles (par exemple, un compte Google ou un identifiant Apple), d'informations personnelles ou d'autres informations sensibles en matière de sécurité lors de vos sessions de test automatique et d'accès à distance.

Appareils privés

Après l'expiration ou la résiliation de votre contrat d'appareil privé, celui-ci ne peut plus être utilisé et est détruit en toute sécurité conformément aux stratégies de destruction AWS. Pour de plus amples informations, veuillez consulter [Appareils privés dans AWS Device Farm](#).

Gestion des clés

Device Farm ne propose actuellement aucune gestion de clé externe pour le chiffrement des données, qu'elles soient au repos ou en transit.

Confidentialité du trafic inter-réseau

Device Farm peut être configuré, pour les appareils privés uniquement, pour utiliser les points de terminaison Amazon VPC pour se connecter à vos ressources. L'accès à toute AWS infrastructure non publique associée à votre compte (par exemple, les EC2 instances Amazon sans adresse IP publique) doit utiliser un point de terminaison Amazon VPC. Quelle que soit la configuration du point de terminaison VPC, Device Farm isole votre trafic des autres utilisateurs du réseau Device Farm.

La sécurité de vos connexions en dehors du AWS réseau n'est pas garantie, et il est de votre responsabilité de sécuriser toutes les connexions Internet établies par vos applications.

Résilience dans AWS Device Farm

L'infrastructure AWS mondiale est construite autour des AWS régions et des zones de disponibilité. Les régions fournissent plusieurs zones de disponibilité physiquement séparées et isolées, connectées par un réseau à faible latence, à haut débit et hautement redondant. Avec les zones de disponibilité, vous pouvez concevoir et exploiter des applications et des bases de données qui basculent automatiquement d'une zone à l'autre sans interruption. Les zones de disponibilité sont davantage disponibles, tolérantes aux pannes et ont une plus grande capacité de mise à l'échelle que les infrastructures traditionnelles à un ou plusieurs centres de données.

Pour plus d'informations sur AWS les régions et les zones de disponibilité, consultez la section [Infrastructure AWS mondiale](#).

Device Farm n'étant disponible que dans la us-west-2 région, nous vous recommandons vivement de mettre en œuvre des processus de sauvegarde et de restauration. Device Farm ne doit pas être la seule source de contenu mis en ligne.

Device Farm ne garantit pas la disponibilité des appareils publics. Ces appareils sont introduits et retirés du parc de périphériques public en fonction de divers facteurs, tels que le taux d'échec et le statut de quarantaine. Nous vous recommandons de ne pas dépendre de la disponibilité d'un seul appareil dans le parc de périphériques public.

Sécurité de l'infrastructure dans AWS Device Farm

En tant que service géré, AWS Device Farm il est protégé par la sécurité du réseau AWS mondial. Pour plus d'informations sur les services AWS de sécurité et sur la manière dont AWS l'infrastructure est protégée, consultez la section [Sécurité du AWS cloud](#). Pour concevoir votre AWS environnement en utilisant les meilleures pratiques en matière de sécurité de l'infrastructure, consultez la section [Protection de l'infrastructure](#) dans le cadre AWS bien architecturé du pilier de sécurité.

Vous utilisez des appels d'API AWS publiés pour accéder à Device Farm via le réseau. Les clients doivent prendre en charge les éléments suivants :

- Protocole TLS (Transport Layer Security). Nous exigeons TLS 1.2 et recommandons TLS 1.3.
- Ses suites de chiffrement PFS (Perfect Forward Secrecy) comme DHE (Ephemeral Diffie-Hellman) ou ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). La plupart des systèmes modernes tels que Java 7 et les versions ultérieures prennent en charge ces modes.

En outre, les demandes doivent être signées à l'aide d'un ID de clé d'accès et d'une clé d'accès secrète associée à un principal IAM. Vous pouvez également utiliser [AWS Security Token Service](#) (AWS STS) pour générer des informations d'identification de sécurité temporaires et signer les demandes.

Sécurité de l'infrastructure pour les tests de périphériques physiques

Pendant les tests physiques, les périphériques sont physiquement séparés. L'isolation du réseau empêche la communication entre les périphériques sur les réseaux sans fil.

Les appareils publics sont partagés, et Device Farm fait de son mieux pour assurer la sécurité des appareils au fil du temps. Certaines actions, telles que les tentatives d'acquisition de droits d'administrateur complets sur un périphérique (pratique appelée débridage ou jailbreak), provoquent la mise en quarantaine de périphériques publics. Ceux-ci sont automatiquement retirés du pool public et font l'objet d'un examen manuel.

Les appareils privés ne sont accessibles que par AWS des comptes explicitement autorisés à le faire. Device Farm isole physiquement ces appareils des autres appareils et les maintient sur un réseau distinct.

Sur les appareils gérés de manière privée, les tests peuvent être configurés pour utiliser un point de terminaison Amazon VPC afin de sécuriser les connexions entrantes et sortantes de votre AWS compte.

Sécurité de l'infrastructure pour les tests de navigateurs de bureau

Lorsque vous utilisez la fonctionnalité de test du navigateur de bureau, toutes les sessions de test sont séparées les unes des autres. Les instances Selenium ne peuvent pas communiquer entre elles sans un tiers intermédiaire, externe à AWS.

Tout le trafic vers les WebDriver contrôleurs Selenium doit être effectué via le point de terminaison HTTPS généré avec `createTestGridUrl`.

Il vous incombe de vous assurer que chaque instance de test Device Farm dispose d'un accès sécurisé aux ressources qu'elle teste. Par défaut, les instances de test du navigateur de bureau de Device Farm ont accès à l'Internet public. Lorsque vous attachez votre instance à un VPC, elle se comporte comme n'importe quelle autre instance EC2, avec un accès aux ressources déterminé par la configuration du VPC et ses composants réseau associés. AWS fournit des [groupes de sécurité et des listes de contrôle d'accès réseau \(ACLs\)](#) pour renforcer la sécurité de votre VPC. Les groupes de sécurité contrôlent le trafic entrant et sortant pour vos ressources, et le réseau ACLs contrôlent le

trafic entrant et sortant pour vos sous-réseaux. Les groupes de sécurité offrent un contrôle d'accès suffisant pour la plupart des sous-réseaux. Vous pouvez utiliser le réseau ACLs si vous souhaitez ajouter une couche de sécurité supplémentaire à votre VPC. Pour obtenir des directives générales sur les meilleures pratiques de sécurité lors de l'utilisation d'Amazon VPCs, consultez [les meilleures pratiques de sécurité](#) pour votre VPC dans le guide de l'utilisateur Amazon Virtual Private Cloud.

Analyse et gestion des vulnérabilités de configuration dans Device Farm

Device Farm vous permet d'exécuter des logiciels qui ne sont pas activement maintenus ou corrigés par le fournisseur, tel que le fournisseur du système d'exploitation, le fournisseur du matériel ou l'opérateur téléphonique. Device Farm fait de son mieux pour maintenir le logiciel à jour, mais ne garantit pas qu'une version particulière du logiciel sur un appareil physique soit à jour, car elle permet de par sa conception de mettre en œuvre des logiciels potentiellement vulnérables.

Par exemple, si un test est effectué sur un appareil exécutant Android 4.4.2, Device Farm ne garantit pas que l'appareil est corrigé contre la [vulnérabilité Android connue sous](#) le nom de. StageFright. Il appartient au fournisseur (et parfois à l'opérateur) du périphérique de fournir les mises à jour de sécurité pour les périphériques. Il n'est pas garanti qu'une application malveillante utilisant cette vulnérabilité soit détectée par notre mise en quarantaine automatisée.

Les appareils privés sont gérés conformément à votre accord avec AWS.

Device Farm met tout en œuvre pour empêcher les applications des clients de commettre des actions telles que le rootage ou le jailbreak. Device Farm supprime les appareils mis en quarantaine du pool public jusqu'à ce qu'ils aient été examinés manuellement.

Vous êtes responsable de la mise à jour de toutes les bibliothèques ou versions de logiciels que vous utilisez dans vos tests, telles que les roues Python et les gemmes Ruby. Device Farm vous recommande de mettre à jour vos bibliothèques de test.

Ces ressources peuvent vous aider à maintenir vos dépendances de test à jour :

- Pour plus d'informations sur la façon de sécuriser les gemmes Ruby, consultez [les pratiques de sécurité](#) sur le RubyGems site Web.
- Pour plus d'informations sur le package de sécurité utilisé par Pipenv et approuvé par la Python Packaging Authority pour analyser votre graphe de dépendances à la recherche de vulnérabilités connues, consultez la section [Détection des vulnérabilités de sécurité](#) sur GitHub.

- Pour plus d'informations sur le vérificateur de dépendances Maven de l'Open Web Application Security Project (OWASP), consultez OWASP sur le site Web de l' [DependencyCheckOWASP](#).

Il est important de se rappeler que même si un système automatisé n'indique pas l'existence de problèmes de sécurité connus, cela ne signifie pas qu'il n'y en a pas. Utilisez toujours avec la prudence raisonnable les bibliothèques ou outils de tiers, et vérifiez les signatures cryptographiques si c'est possible ou raisonnable.

Réponse aux incidents dans Device Farm

Device Farm surveille en permanence les appareils pour détecter les comportements susceptibles d'indiquer des problèmes de sécurité. Si un autre client AWS est informé d'un cas où les données d'un client, telles que les résultats de tests ou les fichiers écrits sur un appareil public, sont accessibles par un autre client, AWS contacte les clients concernés, conformément aux politiques standard d'alerte et de signalement des incidents utilisées dans l'ensemble AWS des services.

Journalisation et surveillance dans Device Farm

Ce service prend AWS CloudTrail en charge un service qui enregistre les AWS appels pour vous Compte AWS et envoie des fichiers journaux à un compartiment Amazon S3. En utilisant les informations collectées par CloudTrail, vous pouvez déterminer à quelles demandes ont été adressées avec succès Services AWS, qui a fait la demande, quand elle a été faite, etc. Pour en savoir plus CloudTrail, notamment sur la façon de l'activer et de trouver vos fichiers journaux, consultez le [guide de AWS CloudTrail l'utilisateur](#).

Pour plus d'informations sur l'utilisation CloudTrail avec Device Farm, consultez [Journalisation des appels d'API AWS Device Farm avec AWS CloudTrail](#).

Bonnes pratiques en matière de sécurité pour Device Farm

Device Farm propose un certain nombre de fonctionnalités de sécurité à prendre en compte lors de l'élaboration et de la mise en œuvre de vos propres politiques de sécurité. Les bonnes pratiques suivantes doivent être considérées comme des instructions générales et ne représentent pas une solution de sécurité complète. Étant donné que ces bonnes pratiques peuvent ne pas être appropriées ou suffisantes pour votre environnement, considérez-les comme des remarques utiles plutôt que comme des recommandations.

- Accordez à tout système d'intégration continue (CI) que vous utilisez le moins de privilèges possible sous IAM. Envisagez d'utiliser des informations d'identification temporaires pour chaque test de système CI, afin que même si un système CI est compromis, il ne puisse pas effectuer de demandes fallacieuses. Pour plus d'informations sur les informations d'identification temporaires, consultez le [guide de l'utilisateur IAM](#).
- Utilisez les commandes adb dans un environnement de test personnalisé pour nettoyer tout contenu créé par votre application. Pour plus d'informations sur les environnements de test personnalisés, consultez [Environnements de test personnalisés](#)

Limites dans AWS Device Farm

Rubriques

- [Service Limits](#)
- [Limites de fichiers](#)
- [Limites d'API](#)
- [Limites des terminaux Appium](#)
- [Limites de variables d'environnement personnalisées](#)

Service Limits

- Il n'existe pas de limite au nombre d'appareils que vous pouvez inclure dans une exécution de test. Cependant, le nombre maximum d'appareils que Device Farm testera simultanément pendant un test est de cinq. Ce nombre peut être augmenté sur demande et évalué au cas par cas par l'équipe de service.
- Il n'y a pas de limite au nombre d'exécutions que vous pouvez planifier. Notez qu'ils ne peuvent rester en file d'attente que pendant 24 heures au maximum.
- La durée d'une session d'accès à distance est strictement limitée à 150 minutes.
- La durée d'un essai automatique est limitée à 150 minutes.
- Le nombre maximum de tâches en cours, y compris les tâches en attente sur votre compte, est de 250. Il s'agit d'une limite souple.
- Il n'y a aucune limite au nombre d'appareils que vous pouvez inclure dans un essai. Le nombre d'appareils (jobs) qui peuvent exécuter vos tests en parallèle à un moment donné est égal à la simultanéité au niveau de votre compte. La simultanéité par défaut au niveau du compte pour une utilisation mesurée dans Device Farm est de cinq.
- La limite de simultanéité mesurée peut être augmentée sur demande jusqu'à un certain seuil en fonction du cas d'utilisation. La simultanéité par défaut au niveau du compte pour une utilisation illimitée est égale au nombre de machines à sous auxquelles vous êtes abonné pour cette plateforme.

Pour plus d'informations concernant les limites de simultanéité mesurées par défaut ou les quotas en général, consultez la page [Quotas](#).

- Une exécution d'automatisation qui n'utilise pas d'[environnement de test personnalisé](#) ne peut contenir que 250 cas de test individuels. Sinon, l'exécution risque d'être ignorée.

Limites de fichiers

- La taille maximale d'un fichier d'application que vous pouvez charger est de 4 Go. Notez que nous n'acceptons actuellement pas les fichiers au format .aab pour Android.
- La taille maximale de la vidéo générée automatiquement par Device Farm lors de votre test est de 1 Go. Toute vidéo dépassant cette taille verra tout le contenu vidéo restant tronqué. Les clients peuvent toujours utiliser leur propre solution d'enregistrement vidéo, le cas échéant, et la stocker en dehors du stockage géré de Device Farm.
- La taille maximale du journal des appareils généré automatiquement par Device Farm (logcat sur Android ou syslog sur iOS) pendant votre test est de 1 Go. Tout journal dépassant cette taille verra tous les journaux restants tronqués. Pour les journaux supérieurs à 1 Go, les clients peuvent les stocker en dehors du stockage géré de Device Farm.
- La taille maximale cumulée des artefacts clients en mode environnement personnalisé de Device Farm est de 1 Go. Si cette taille est dépassée par vos artefacts, aucun des artefacts ne sera disponible.
- Si la taille cumulée de tous les artefacts générés lors d'un test dépasse 4 Go, certains artefacts peuvent être supprimés (notamment la vidéo, les journaux de l'appareil et les artefacts du client).

Limites d'API

- Device Farm utilise un algorithme tokenbucket pour limiter le taux d'appels des API. Par exemple, imaginez la création d'un bucket contenant des jetons. Chaque jeton représente une transaction, et un appel d'API utilise un jeton. Les jetons sont ajoutés au compartiment à un taux fixe (par exemple, 10 jetons par seconde), et le compartiment a une capacité maximale (par exemple, 100 jetons). Lorsqu'une demande ou un paquet arrive, il doit récupérer un jeton dans le compartiment à traiter. S'il y a suffisamment de jetons, la demande est autorisée et les jetons sont supprimés. S'il n'y a pas assez de jetons, la demande est retardée ou supprimée, selon l'implémentation.

Dans Device Farm, voici comment l'algorithme est implémenté :

- Les demandes d'API Burst sont le nombre maximum de demandes auxquelles le service est capable de répondre pour une API spécifiée dans un identifiant de compte client spécifié. En

d'autres termes, il s'agit de la capacité du godet. Vous pouvez appeler l'API autant de fois qu'il reste des jetons dans le compartiment, et chaque demande consomme un jeton.

- Le taux Transactions-per-second (TPS) est le taux minimum auquel vos demandes d'API peuvent être exécutées. En d'autres termes, il s'agit du taux auquel le seau se recharge en jetons par seconde. Par exemple, si une API possède un numéro de rafale de dix mais un TPS de un, vous pouvez l'appeler dix fois instantanément. Cependant, le bucket ne récupérerait des jetons qu'au rythme d'un jeton par seconde, ce qui se traduirait par un appel par seconde, sauf si vous arrêtez d'appeler l'API pour laisser le bucket se remplir.

Voici les tarifs de Device Farm APIs :

- Pour List and Get APIs, la capacité des requêtes de l'API Burst est de 50, et le taux Transactions-per-second (TPS) est **10** de.
- Pour tous les autres APIs, la capacité des requêtes de l'API Burst est de 10, et le taux Transactions-per-second (TPS) est **1** de.

Limites des terminaux Appium

Les limites suivantes s'appliquent à toutes les sessions de point de terminaison Appium. Pour toute question ou pour obtenir des conseils sur la meilleure façon de gérer les limites, veuillez ouvrir un dossier d'assistance.

- Chaque commande Appium a une durée d'exécution maximale de 4 minutes, après quoi la commande expire.
- Le point de terminaison accepte des tailles de charge utile d'entrée allant jusqu'à 20 Mo et autorise des tailles de charge utile de sortie allant jusqu'à 20 Mo. Toute demande dont la taille d'entrée ou de sortie est supérieure à celle-ci recevra une WebDriver erreur de 'unsupported operation'.
- Les demandes s'exécutent séquentiellement sur l'appareil dans l'ordre dans lequel elles sont reçues. Par conséquent, nous vous recommandons vivement d'envoyer les commandes de manière séquentielle et d'attendre la réponse de chaque commande avant d'en envoyer une nouvelle. Cela dit, certaines commandes du serveur Appium peuvent être envoyées en parallèle, notamment :
 - [Obtenir le statut](#)
 - [Obtenir des sessions](#)
- Le point de terminaison ne prend pas en charge le [WebDriver BiDi protocole](#) pour le moment.

- Le point de terminaison ne prend pas en charge les plugins ou pilotes Appium autres que les pilotes XCUITest and UIAutomator2 .
- Un maximum de 3 applications peuvent être utilisées comme applications auxiliaires avec une demande de création de session d'accès à distance. Cela dit, il n'y a aucune limite quant au nombre d'applications pouvant être installées au cours d'une session à l'aide de l'[InstallToRemoteAccessSession](#)API.

Limites de variables d'environnement personnalisées

Les limites suivantes s'appliquent à toutes les variables d'environnement personnalisées. Pour toute question ou pour obtenir des conseils sur la meilleure façon de gérer les limites, veuillez ouvrir un dossier d'assistance.

- Un maximum de 32 variables peuvent être configurées ou exécutées sur un projet Device Farm donné.
- Les noms de variables ne peuvent pas dépasser 256 caractères.
- Les noms de variables sont soumis aux limites imposées par bash. À savoir, ils ne doivent contenir que des caractères alphanumériques et des traits de soulignement, et ne peuvent pas commencer par un chiffre.
- Les noms de variables commençant par \$DEVICEFARM_ sont réservés à un usage interne du service.
- Les valeurs des variables ne peuvent pas dépasser 256 caractères.
- Les variables d'environnement ne peuvent pas être utilisées pour configurer la sélection du calcul de l'hôte de test dans le fichier de spécifications de test.

Outils et plug-ins pour AWS Device Farm

Cette section contient des liens et des informations sur l'utilisation des outils et plug-ins AWS Device Farm. Vous pouvez trouver les plug-ins Device Farm sur [AWS Labs à l'adresse GitHub](#).

Si vous êtes un développeur Android, nous avons également un [exemple d'application AWS Device Farm pour Android sur GitHub](#). Vous pouvez utiliser l'application et des exemples de tests comme référence pour vos propres scripts de test Device Farm.

Rubriques

- [Intégration de Device Farm à un serveur Jenkins CI](#)
- [Intégrer Device Farm à un système de construction Gradle](#)

Intégration de Device Farm à un serveur Jenkins CI

Le plugin Jenkins CI fournit les fonctionnalités d'AWS Device Farm à partir de votre propre serveur d'intégration continue (CI) Jenkins. Pour plus d'informations, consultez [Jenkins \(logiciel\)](#).

Note

Pour télécharger le plugin Jenkins, rendez-vous sur [GitHub](#) et suivez les instructions dans [Étape 1 : Installation du plugin Jenkins CI pour AWS Device Farm](#).

Cette section contient une série de procédures pour configurer et utiliser le plug-in Jenkins CI avec AWS Device Farm.

Les images suivantes montrent les fonctions du plug-in Jenkins CI.



- [Back to Dashboard](#)
- [Status](#)
- [Changes](#)
- [Workspace](#)
- [Build Now](#)
- [Delete Project](#)
- [Configure](#)
- [AWS Device Farm](#)

Project Hello World App

- [Workspace](#)
- [Recent Changes](#)

Build History	trend
#19 Jul 15, 2015 4:25 AM	
#18 Jul 15, 2015 1:35 AM	
#17 Jul 15, 2015 1:21 AM	
#16 Jul 15, 2015 1:06 AM	
#15 Jul 14, 2015 10:55 PM	
RSS for all RSS for failures	



Recent AWS Device Farm Results

Status	Build Number	Pass/Warn/Skip/Fail/Error/Stop							Web Report
Completed	#19	12 	0 	1 	1 	1 	0 	Full Report	
Completed	#18	9 	0 	1 	1 	1 	0 	Full Report	
Completed	#17	12 	0 	1 	1 	1 	0 	Full Report	
Completed	#16	12 	0 	1 	1 	1 	0 	Full Report	
Completed	#15	11 	0 	1 	2 	1 	0 	Full Report	

Permalinks

- [Last build \(#19\), 41 min ago](#)
- [Last failed build \(#19\), 41 min ago](#)
- [Last unsuccessful build \(#19\), 41 min ago](#)

Post-build Actions

Run Tests on AWS Device Farm

[refresh](#)

Project ?

[Required] Select your AWS Device Farm project.

Device Pool ?

[Required] Select your AWS Device Farm device pool.

Application ?

[Required] Pattern to find newly built application.

☐ Store test results locally.

Choose test to run

- ☐ Built-in Fuzz
- ☐ Appium Java JUnit
- ☐ Appium Java TestNG
- ☒ Calabash

Features ?

[Required] Pattern to find features.zip.

Tags ?

[Optional] Tags to pass into Calabash.

- ☐ Instrumentation
- ☐ Android UI Automator

[Delete](#)[Add post-build action](#) ▼[Save](#)[Apply](#)

Le plug-in peut également afficher tous les artefacts de test (journaux, captures d'écran, etc.) localement :



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 \(WiFi\)](#)

 [\(all files in zip\)](#)

Navigation:

- [Back to Project](#)
- [Status](#)
- [Changes](#)
- [Console Output](#)
- [Edit Build Information](#)
- [Delete Build](#)
- [AWS Device Farm](#)
- [Previous Build](#)

Rubriques

- [Dépendances](#)
- [Étape 1 : Installation du plugin Jenkins CI pour AWS Device Farm](#)
- [Étape 2 : Création d'un Gestion des identités et des accès AWS utilisateur pour votre plug-in Jenkins CI pour AWS Device Farm](#)
- [Étape 3 : Configuration du plugin Jenkins CI pour la première fois dans AWS Device Farm](#)
- [Étape 4 : Utilisation du plugin dans une tâche Jenkins](#)

Dépendances

Le plug-in Jenkins CI nécessite le SDK AWS mobile 1.10.5 ou version ultérieure. Pour plus d'informations et pour installer le kit SDK, consultez [Kit SDK AWS Mobile](#).

Étape 1 : Installation du plugin Jenkins CI pour AWS Device Farm

Il existe deux options pour installer le plug-in d'intégration continue (CI) Jenkins pour AWS Device Farm. Vous pouvez rechercher le plug-in à partir de la boîte de dialogue Available Plugins (Plug-ins disponibles) de l'interface utilisateur Web Jenkins, ou vous pouvez télécharger le fichier `hpi` et l'installer à partir de Jenkins.

Installer à partir de l'interface utilisateur Jenkins

1. Recherchez le plug-in dans l'interface utilisateur Jenkins en choisissant Manage Jenkins (Gérer Jenkins), Manage Plugins (Gérer les plug-ins), puis Available (Disponible).
2. Recherchez aws-device-farm.
3. Installez le plugin AWS Device Farm.
4. Assurez-vous que le plug-in est détenu par l'utilisateur Jenkins.
5. Redémarrez Jenkins.

Téléchargez le plugin

1. Téléchargez le hpi fichier directement depuis <http://updates.jenkins-ci.org/latest/aws-device-farm.hpi>.
2. Assurez-vous que le plug-in est détenu par l'utilisateur Jenkins.
3. Installez le plug-in à l'aide d'une des options suivantes :
 - Chargez le plug-in en choisissant Manage Jenkins (Gérer Jenkins), Manage Plugins (Gérer les plug-ins), Advanced (Avancé), puis Upload plugin (Charger le plug-in).
 - Placez le fichier hpi dans le répertoire de plug-ins Jenkins (généralement `/var/lib/jenkins/plugins`).
4. Redémarrez Jenkins.

Étape 2 : Création d'un Gestion des identités et des accès AWS utilisateur pour votre plug-in Jenkins CI pour AWS Device Farm

Nous vous recommandons de ne pas utiliser votre compte AWS root pour accéder à Device Farm. Créez plutôt un nouvel utilisateur Gestion des identités et des accès AWS (IAM) (ou utilisez un utilisateur IAM existant) dans votre AWS compte, puis accédez à Device Farm avec cet utilisateur IAM.

Pour créer un nouvel utilisateur IAM, voir [Création d'un utilisateur IAM \(\)AWS Management Console](#). Veillez à générer une clé d'accès pour chaque utilisateur et téléchargez ou enregistrez les informations d'identification de sécurité de l'utilisateur. Vous aurez besoin des informations d'identification de l'utilisateur ultérieurement.

Donnez à l'utilisateur IAM l'autorisation d'accéder à Device Farm

Pour autoriser l'utilisateur IAM à accéder à Device Farm, créez une nouvelle politique d'accès dans IAM, puis attribuez-la à l'utilisateur IAM comme suit.

Note

Le compte AWS root ou l'utilisateur IAM que vous utilisez pour effectuer les étapes suivantes doit être autorisé à créer la politique IAM suivante et à l'associer à l'utilisateur IAM. Pour plus d'informations, consultez [Utilisation de stratégies](#).

Pour créer la politique d'accès dans IAM

1. Ouvrez la console IAM à l'adresse <https://console.aws.amazon.com/iam/>.
2. Choisissez Politiques (Politiques).
3. Choisissez Create Policy (Créer une politique). (Si un bouton Mise en route est affiché, choisissez-le, puis choisissez Créer une stratégie.)
4. En regard de Create Your Own Policy (Créez votre politique), choisissez Select (Sélectionner).
5. Pour Nom de la stratégie, saisissez un nom de stratégie (par exemple **AWSDeviceFarmAccessPolicy**).
6. Dans Description, tapez une description qui vous aidera à associer cet utilisateur IAM à votre projet Jenkins.
7. Pour Document de stratégie, saisissez l'instruction suivante :

JSON

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

```
}
```

8. Choisissez Create Policy (Créer une politique).

Pour attribuer la politique d'accès à l'utilisateur IAM

1. Ouvrez la console IAM à l'adresse <https://console.aws.amazon.com/iam/>.
2. Choisissez Utilisateurs.
3. Choisissez l'utilisateur IAM à qui vous allez attribuer la politique d'accès.
4. Dans la zone Autorisations, pour Stratégies gérées, choisissez Attacher la stratégie.
5. Sélectionnez la stratégie que vous venez de créer (par exemple AWSDeviceFarmAccessPolicy).
6. Choisissez Attach Policy (Attacher une politique).

Étape 3 : Configuration du plugin Jenkins CI pour la première fois dans AWS Device Farm

La première fois que vous exécutez votre serveur Jenkins, vous devez configurer le système comme suit.

Note

Si vous utilisez des [emplacements d'appareils](#), la fonction d'emplacement d'appareil est désactivée par défaut.

1. Connectez-vous à votre interface utilisateur Web Jenkins.
2. À gauche de l'écran, choisissez Manage Jenkins (Gérer Jenkins).
3. Choisissez Configure System (Configurer le système).
4. Faites défiler la page jusqu'à l'en-tête AWS Device Farm.
5. Copiez vos identifiants de sécurité de votre [Création d'un utilisateur IAM pour votre plugin Jenkins CI](#) et collez votre ID de clé d'accès et votre clé d'accès secrète dans leurs zones respectives.
6. Choisissez Enregistrer.

Étape 4 : Utilisation du plugin dans une tâche Jenkins

Une fois que vous avez installé le plug-in Jenkins, suivez ces instructions pour utiliser le plug-in dans une tâche Jenkins.

1. Connectez-vous à votre interface utilisateur Web Jenkins.
2. Cliquez sur la tâche que vous souhaitez modifier.
3. À gauche de l'écran, choisissez Configure (Configurer).
4. Faites défiler l'écran jusqu'à l'en-tête Post-build Actions (Actions de post-production).
5. Cliquez sur Ajouter une action post-build et sélectionnez Run Tests on AWS Device Farm.
6. Sélectionnez le projet que vous souhaitez utiliser.
7. Sélectionnez le groupe d'appareils que vous souhaitez utiliser.
8. Indiquez si vous souhaitez obtenir les artefacts de test (telles que les journaux et les captures d'écran) archivés localement.
9. Dans Application, indiquez le chemin de votre application compilée.
10. Sélectionnez le test que vous souhaitez exécuter et renseignez tous les champs obligatoires.
11. Choisissez Enregistrer.

Intégrer Device Farm à un système de construction Gradle

Le plugin Device Farm Gradle permet l'intégration d'AWS Device Farm au système de build Gradle dans Android Studio. Pour plus d'informations, consultez [Gradle](#).

Note

Pour télécharger le plugin Gradle, rendez-vous sur [GitHub](#) et suivez les instructions dans [Création du plugin Device Farm Gradle](#).

Le plugin Device Farm Gradle fournit les fonctionnalités Device Farm à partir de votre environnement Android Studio. Vous pouvez lancer les tests sur de vrais téléphones et tablettes Android hébergés par Device Farm.

Cette section contient une série de procédures pour configurer et utiliser le plugin Device Farm Gradle.

Rubriques

- [Dépendances](#)
- [Étape 1 : Création du plugin AWS Device Farm Gradle](#)
- [Étape 2 : Configuration du plugin AWS Device Farm Gradle](#)
- [Étape 3 : Génération d'un utilisateur IAM dans le plugin Device Farm Gradle](#)
- [Étape 4 : Configuration des types de tests](#)

Dépendances

Exécution

- Le plug-in Device Farm Gradle nécessite le SDK AWS mobile 1.10.15 ou version ultérieure. Pour plus d'informations et pour installer le kit SDK, consultez [Kit SDK AWS Mobile](#).
- Android tools builder test api 0.5.2
- Apache Commons Lang3 3.3.4

Pour les tests d'unité

- Testng 6.8.8
- Jmockit 1.19
- Android gradle tools 1.3.0

Étape 1 : Création du plugin AWS Device Farm Gradle

Ce plugin permet l'intégration d'AWS Device Farm au système de build Gradle dans Android Studio. Pour plus d'informations, consultez [Gradle](#).

Note

La création du plug-in est facultative. Le plug-in est publié via Maven Central. Si vous souhaitez autoriser Gradle à télécharger le plug-in directement, ignorez cette étape et passez à [Étape 2 : Configuration du plugin AWS Device Farm Gradle](#).

Pour créer le plug-in

1. Accédez au dépôt [GitHub](#) et clonez-le.
2. Créez le plug-in à l'aide de gradle `install`.

Le plug-in est installé sur votre référentiel Maven local.

Étape suivante : [Étape 2 : Configuration du plugin AWS Device Farm Gradle](#)

Étape 2 : Configuration du plugin AWS Device Farm Gradle

Si ce n'est pas déjà fait, clonez le référentiel et installez le plug-in à l'aide de la procédure suivante : [Création du plugin Device Farm Gradle](#).

Pour configurer le plug-in AWS Device Farm Gradle

1. Ajoutez l'artéfact du plug-in à votre liste de dépendances dans `build.gradle`.

```
buildscript {  
  
    repositories {  
        mavenLocal()  
        mavenCentral()  
    }  
  
    dependencies {  
        classpath 'com.android.tools.build:gradle:1.3.0'  
        classpath 'com.amazonaws:aws-devicefarm-gradle-plugin:1.0'  
    }  
}
```

2. Configurez le plug-in dans votre fichier `build.gradle`. Le configuration spécifique au test suivante doit vous servir de guide :

```
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. Exécutez votre test Device Farm à l'aide de la tâche suivante :`gradle devicefarmUpload`.

La sortie de compilation affichera un lien vers la console Device Farm où vous pourrez surveiller l'exécution de vos tests.

Étape suivante : [Génération d'un utilisateur IAM dans le plugin Device Farm Gradle](#)

Étape 3 : Génération d'un utilisateur IAM dans le plugin Device Farm Gradle

Gestion des identités et des accès AWS (IAM) vous aide à gérer les autorisations et les politiques relatives à l'utilisation des AWS ressources. Cette rubrique explique comment créer un utilisateur IAM autorisé à accéder aux ressources AWS Device Farm.

Si ce n'est pas déjà fait, effectuez les étapes 1 et 2 avant de générer un utilisateur IAM.

Nous vous recommandons de ne pas utiliser votre compte AWS root pour accéder à Device Farm. Créez plutôt un nouvel utilisateur IAM (ou utilisez un utilisateur IAM existant) dans votre AWS compte, puis accédez à Device Farm avec cet utilisateur IAM.

 Note

Le compte AWS root ou l'utilisateur IAM que vous utilisez pour effectuer les étapes suivantes doit être autorisé à créer la politique IAM suivante et à l'associer à l'utilisateur IAM. Pour plus d'informations, consultez [Utilisation de stratégies](#).

Pour créer un nouvel utilisateur avec la politique d'accès appropriée dans IAM

1. Ouvrez la console IAM à l'adresse <https://console.aws.amazon.com/iam/>.
2. Choisissez Utilisateurs.
3. Choisissez Créer des utilisateurs.
4. Entrez le nom d'utilisateur de votre choix.

Par exemple, **GradleUser**.

5. Choisissez Créer.
6. Choisissez Télécharger les informations d'identification et enregistrez-les à un emplacement où vous pourrez facilement les récupérer ultérieurement.
7. Choisissez Fermer.
8. Choisissez le nom d'utilisateur dans la liste.
9. Sous Autorisations, développez l'en-tête Stratégies en ligne en cliquant sur la flèche vers le bas située à droite.
10. Choisissez Cliquez ici là où il est écrit « Il n'y a aucune politique en ligne à afficher ». Pour en créer un, cliquez ici.
11. Sur l'écran Set Permissions (Réglez les permissions), choisissez Stratégie personnalisée.
12. Choisissez Select (Sélectionner).
13. Attribuez un nom à votre stratégie, par exemple **AWSDeviceFarmGradlePolicy**.
14. Collez la stratégie suivante dans Document de stratégie.

JSON

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

```
{
  "Sid": "DeviceFarmAll",
  "Effect": "Allow",
  "Action": [ "devicefarm:*" ],
  "Resource": [ "*" ]
}
```

15. Choisissez Apply Policy (Appliquer la stratégie).

Étape suivante: [Configuration des types de tests](#).

Pour plus d'informations, consultez [Création d'un utilisateur IAM \(AWS Management Console\)](#) ou [Configuration](#).

Étape 4 : Configuration des types de tests

Par défaut, le plug-in AWS Device Farm Gradle exécute le [Instrumentation pour Android et AWS Device Farm](#) test. Si vous souhaitez exécuter vos propres tests ou spécifier des paramètres supplémentaires, vous pouvez choisir de configurer un type de test. Cette rubrique fournit des informations sur chaque type de test disponible, ainsi que la procédure à suivre dans Android Studio pour le configurer afin de l'utiliser. Pour plus d'informations sur les types de tests disponibles dans Device Farm, consultez [Frameworks de test et tests intégrés dans AWS Device Farm](#).

Si ce n'est pas déjà fait, effectuez les étapes 1 à 3 avant de configurer les types de tests.

Note

Si vous utilisez des [emplacements d'appareils](#), la fonction d'emplacement d'appareil est désactivée par défaut.

Appium

Device Farm prend en charge Appium Java et JUnit TestNG pour Android.

- [Appium \(sous Java \(\)\) JUnit](#)
- [Appium \(sous Java \(TestNG\)\)](#)

Vous pouvez choisir `useTestNG()` ou `useJUnit()`. `JUnit` est la valeur par défaut et n'a pas besoin d'être explicitement spécifié.

```
appium {
    tests file("path to zip file") // required
    useTestNG() // or useJUnit()
}
```

Intégré : fuzz

Device Farm propose un type de test de fuzz intégré, qui envoie de manière aléatoire les événements de l'interface utilisateur aux appareils, puis communique les résultats.

```
fuzz {

    eventThrottle 50 // optional default
    eventCount 6000 // optional default
    randomizerSeed 1234 // optional default blank

}
```

Pour de plus amples informations, veuillez consulter [Exécution du test de fuzz intégré à Device Farm \(Android et iOS\)](#).

Instrumentation

Device Farm prend en charge l'instrumentation (EspressoJUnit, Robotium ou tout autre test basé sur l'instrumentation) pour Android. Pour de plus amples informations, veuillez consulter [Instrumentation pour Android et AWS Device Farm](#).

Lorsque vous exécutez un test d'instrumentation dans Gradle, Device Farm utilise le `.apk` fichier généré à partir de votre répertoire `AndroidTest` comme source de vos tests.

```
instrumentation {

    filter "test filter per developer docs" // optional

}
```

Historique du document AWS Device Farm

Le tableau suivant décrit les modifications importantes apportées à la documentation depuis la dernière version de ce manuel.

Modifier	Description	Date de modification
Support des terminaux Appium	Device Farm propose désormais un point de terminaison Appium entièrement géré pour les tests à distance des appareils, ce qui permet un développement et un débogage rapides des tests. Cela complète la méthode d'exécution côté serveur existante, dans laquelle les tests sont téléchargés et exécutés directement sur Device Farm. Alors que l'exécution côté serveur est idéale pour les CI/CD pipelines et les tests à grande échelle, le nouveau point de terminaison Appium local permet d'accélérer l'itération et le développement de tests sur des appareils réels.	17 novembre 2025
Améliorations apportées à l'hôte de test iOS	Device Farm prend désormais en charge une expérience actualisée pour l'environnement de test iOS, ce qui permet de garantir la cohérence des configurations entre les tests Android et iOS. Pour en savoir plus, consultez Hôtes pour environnements de test personnalisés . En outre, les informations relatives aux hôtes de test Android retirés ont été supprimées. Les utilisateurs d'Android sont invités à utiliser les hôtes de test Amazon Linux 2 .	31 octobre 2025
AL2 soutien	Device Farm prend désormais en charge l'environnement de AL2 test pour Android. En savoir plus sur AL2 .	6 novembre 2023
Migration d'environnements de test standard vers des	Guide de migration mis à jour pour documenter la dépréciation des documents pour les tests en mode standard en décembre 2023.	3 septembre 2023

Modifier	Description	Date de modification
environnements de test personnalisés		
Support VPC ENI	Device Farm permet désormais aux appareils privés d'utiliser la fonctionnalité de connectivité VPC-ENI pour aider les clients à se connecter en toute sécurité à leurs points de terminaison privés hébergés sur AWS, un logiciel sur site ou un autre fournisseur de cloud. En savoir plus sur VPC-ENI .	15 mai 2023
mises à jour de l'interface utilisateur Polaris	La console Device Farm prend désormais en charge le framework Polaris.	28 juillet 2021
Prise en charge de Python 3	Device Farm prend désormais en charge Python 3 dans les tests en mode personnalisé. En savoir plus sur l'utilisation de Python 3 dans vos packages de test : • Appium (Python) • Appium (Python)	20 avril 2020
Nouvelles informations de sécurité et informations sur le balisage des AWS ressources.	Pour rendre la sécurisation AWS des services plus simple et plus complète, une nouvelle section sur la sécurité a été créée. Pour en savoir plus, consultez Sécurité dans AWS Device Farm Une nouvelle section sur le balisage dans Device Farm a été ajoutée. Pour plus d'informations sur le balisage, consultez Marquage dans Device Farm.	27 mars 2020
Suppression de l'accès direct aux appareils.	L'accès direct aux périphériques (débogage à distance sur des périphériques privés) n'est plus disponible pour une utilisation générale. Pour toute question relative à la future disponibilité de l'accès direct aux périphériques, contactez-nous .	9 septembre 2019

Modifier	Description	Date de modification
Mise à jour de la configuration du plug-in Gradle	Une configuration de plug-in Gradle révisée inclut désormais une version personnalisable de la configuration Gradle, avec des paramètres facultatifs mis en commentaire. En savoir plus sur Configuration du plugin Device Farm Gradle .	16 août 2019
Nouvelle exigence pour les essais avec XCTest	Pour les tests utilisant le XCTest framework, Device Farm nécessite désormais un package d'application conçu pour les tests. En savoir plus sur the section called "XCTest" .	4 février 2019
Prise en charge des types de test Appium Node.js et Ruby dans des environnements personnalisés	Vous pouvez désormais exécuter vos tests dans les environnements de test personnalisés Appium Node.js et Appium Ruby. En savoir plus sur Frameworks de test et tests intégrés dans AWS Device Farm .	10 janvier 2019
Prise en charge de la version 1.7.2 du serveur Appium dans les environnements standard et personnalisés. Prise en charge de la version 1.8.1 en utilisant un fichier YAML de spécification de test personnalisé dans un environnement de test personnalisé.	Vous pouvez désormais exécuter vos tests dans les environnements de test standard et personnalisés avec les versions 1.7.2, 1.7.1 et 1.6.5 du serveur Appium. Vous pouvez également exécuter vos tests avec les versions 1.8.1 et 1.8.0 en utilisant un fichier YAML de spécification de test personnalisé dans un environnement de test personnalisé. En savoir plus sur Frameworks de test et tests intégrés dans AWS Device Farm .	2 octobre 2018

Modifier	Description	Date de modification
Environnements de test personnalisés	Avec un environnement de test personnalisé, vous pouvez vous assurer que vos tests s'exécutent comme ils le font dans votre environnement local. Device Farm prend désormais en charge les journaux en direct et le streaming vidéo, afin que vous puissiez obtenir des commentaires instantanés sur vos tests exécutés dans un environnement de test personnalisé. En savoir plus sur Environnements de test personnalisés dans AWS Device Farm .	16 août 2018
Support pour l'utilisation de Device Farm en tant que fournisseur de AWS CodePipeline tests	Vous pouvez désormais configurer un pipeline AWS CodePipeline pour utiliser les runs d'AWS Device Farm comme actions de test dans votre processus de publication. CodePipeline vous permet de relier rapidement votre référentiel aux étapes de création et de test afin de créer un système d'intégration continue adapté à vos besoins. En savoir plus sur Intégration d'AWS Device Farm dans une phase CodePipeline de test .	19 juillet 2018
Prise en charge des appareils privés	Vous pouvez désormais utiliser des appareils privés pour planifier des séries de tests et démarrer des sessions d'accès à distance. Vous pouvez gérer les profils et les paramètres de ces appareils, créer des points de terminaison Amazon VPC pour tester des applications privées et créer des sessions de débogage à distance. En savoir plus sur Appareils privés dans AWS Device Farm .	2 mai 2018
Prise en charge d'Appium 1.6.3	Vous pouvez désormais définir la version d'Appium pour vos tests personnalisés Appium.	21 mars 2017
Définir le délai d'exécution des séries de tests	Vous pouvez définir le délai d'exécution d'une série de tests ou pour tous les tests d'un projet. En savoir plus sur Définition du délai d'exécution pour les tests dans AWS Device Farm .	9 février 2017

Modifier	Description	Date de modification
Mise en forme réseau	Vous pouvez désormais simuler des connexions et des conditions réseau pour une série de tests. En savoir plus sur Simulation des connexions réseau et des conditions pour les exécutions de votre AWS Device Farm .	8 décembre 2016
Nouvelle section de résolution des problèmes	Vous pouvez désormais résoudre les problèmes liés aux téléchargements de packages de test à l'aide d'un ensemble de procédures conçues pour résoudre les messages d'erreur que vous pourriez rencontrer dans la console Device Farm. En savoir plus sur Résolution des erreurs liées à Device Farm .	10 août 2016
Sessions d'accès distant	Vous pouvez désormais accéder à distance et interagir avec un seul appareil dans la console. En savoir plus sur Accès à distance .	19 avril 2016
Libre-service pour les emplacements d'appareils	Vous pouvez désormais acheter des emplacements pour appareils à l'aide de l'API AWS Management Console AWS Command Line Interface, du ou de l'API. Découvrez comment Acheter un emplacement pour appareil dans Device Farm .	22 mars 2016
Comment arrêter les exécutions de test	Vous pouvez désormais arrêter les tests à l'aide de l' AWS Management Console API AWS Command Line Interface, de ou de l'API. Découvrez comment Arrêter une exécution dans AWS Device Farm .	22 mars 2016
Nouveaux types de tests d' XCTest interface utilisateur	Vous pouvez désormais exécuter des tests personnalisés de l' XCTest interface utilisateur sur les applications iOS. En savoir plus sur les types de tests Intégration de l' XCTest interface utilisateur pour iOS à Device Farm .	8 mars 2016

Modifier	Description	Date de modification
Nouveaux types de tests Appium écrits en Python	Vous pouvez désormais exécuter des tests personnalisés Appium Python sur des applications Android, iOS et Web. En savoir plus sur Frameworks de test et tests intégrés dans AWS Device Farm .	19 janvier 2016
Types de tests d'application Web	Vous pouvez désormais exécuter des tests personnalisés Appium Java JUnit et TestNG sur des applications Web. En savoir plus sur Tests d'applications Web dans AWS Device Farm .	19 novembre 2015
Plug-in AWS Device Farm Gradle	Découvrez comment installer et utiliser le Plug-in Device Farm Gradle .	28 septembre 2015
Nouveau test Android intégré : explorateur	Le test d'explorateur analyse chaque écran de votre application comme s'il était un utilisateur final et effectue des captures d'écran à mesure qu'il explore.	16 septembre 2015
Ajout de la prise en charge d'iOS	En savoir plus sur le test des appareils iOS et l'exécution de tests iOS (y compris XCTest) dans Frameworks de test et tests intégrés dans AWS Device Farm .	4 août 2015
Première version publique	Il s'agit de la première version publique du manuel AWS Device Farm Developer Guide.	13 juillet 2015

AWS Glossaire

Pour la AWS terminologie la plus récente, consultez le [AWS glossaire](#) dans la Glossaire AWS référence.

Les traductions sont fournies par des outils de traduction automatique. En cas de conflit entre le contenu d'une traduction et celui de la version originale en anglais, la version anglaise prévaudra.