



Guide du développeur

AWS Device Farm



Version de l'API 2015-06-23

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

AWS Device Farm: Guide du développeur

Copyright © 2024 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 |
| Tests automatisés des applications | 1 |
| Interaction d'accès à distance | 1 |
| 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 | 6 |
| Démarrer | 7 |
| Prérequis | 7 |
| Étape 1 : Se connecter à la console | 8 |
| Étape 2 : Création d'un projet | 8 |
| Étape 3 : créer et démarrer une course | 8 |
| Étape 4 : Afficher les résultats de la course | 10 |
| Étapes suivantes | 11 |
| Achetez des emplacements pour appareils | 12 |
| Acheter des emplacements pour appareils (console) | 12 |
| Achetez un emplacement pour appareil (AWS CLI) | 14 |
| Acheter un emplacement pour appareil (API) | 18 |
| Concepts | 20 |
| Appareils | 20 |
| Appareils pris en charge | 20 |
| Pools d'appareils | 21 |
| Appareils privés | 21 |
| Identification de l'appareil | 21 |
| Fentes pour appareils | 21 |
| Applications préinstallées pour appareils | 22 |
| Fonctionnalités de l'appareil | 22 |
| Environnements de test | 22 |
| Environnement de test standard | 23 |
| Environnement de test personnalisé | 23 |
| Exécutions | 23 |

| | |
|---|----|
| Exécuter la configuration | 24 |
| Exécuter la conservation des fichiers | 24 |
| Exécuter l'état de l'appareil | 24 |
| Sorties en parallèle | 25 |
| Configuration du délai d'exécution | 25 |
| Applications d'instrumentation | 25 |
| Resignature d'applications en plusieurs fois | 25 |
| Applications obfusquées en cours d'exécution | 25 |
| Publicités en séries | 26 |
| Médias en course | 26 |
| Tâches courantes pour les courses | 26 |
| Rapports | 26 |
| Conservation des rapports | 26 |
| Composants du rapport | 26 |
| Se connecte aux rapports | 27 |
| Tâches courantes pour les rapports | 27 |
| Séances | 27 |
| Appareils compatibles pour l'accès à distance | 27 |
| Conservation des fichiers de session | 27 |
| Applications d'instrumentation | 28 |
| Resignature d'applications dans les sessions | 28 |
| Applications masquées dans les sessions | 28 |
| Travailler avec des projets | 29 |
| Créer un projet | 29 |
| Prérequis | 29 |
| Création d'un projet (console) | 29 |
| Créer un projet (AWS CLI) | 30 |
| Création d'un projet (API) | 30 |
| Voir la liste des projets | 30 |
| Prérequis | 31 |
| Afficher la liste des projets (console) | 31 |
| Consultez la liste des projets (AWS CLI) | 31 |
| Afficher la liste des projets (API) | 31 |
| Travailler avec des essais | 32 |
| Création d'un essai | 32 |
| Prérequis | 33 |

| | |
|---|----|
| Création d'un test (console) | 33 |
| Créer un test (AWS CLI) | 36 |
| Création d'un test (API) | 46 |
| Étapes suivantes | 47 |
| Définir le délai d'exécution | 47 |
| Prérequis | 48 |
| Définir le délai d'exécution d'un projet | 48 |
| Définissez le délai d'exécution d'un test | 49 |
| Simulez les connexions et les conditions du réseau | 49 |
| Configurer le modelage du réseau lors de la planification d'un test | 50 |
| Création d'un profil réseau | 50 |
| Modifier les conditions du réseau pendant votre test | 52 |
| Arrêtez une course | 52 |
| Arrêter une exécution (console) | 52 |
| Arrêtez une course (AWS CLI) | 54 |
| Arrêter une exécution (API) | 56 |
| Afficher la liste des courses | 56 |
| Afficher la liste des exécutions (console) | 56 |
| Afficher la liste des courses (AWS CLI) | 56 |
| Afficher la liste des exécutions (API) | 57 |
| Créer un groupe d'appareils | 57 |
| Prérequis | 57 |
| Création d'un pool d'appareils (console) | 57 |
| Création d'un pool d'appareils (AWS CLI) | 59 |
| Création d'un pool d'appareils (API) | 59 |
| Analyse des résultats | 59 |
| Utilisation des rapports de test | 59 |
| Travailler avec des artefacts | 69 |
| Marquage dans Device Farm | 75 |
| Étiquetage des ressources | 75 |
| Recherche de ressources par tag | 76 |
| Suppression de balises de ressources | 77 |
| Types de tests et cadres | 78 |
| Cadres de test | 78 |
| Cadres de test d'applications Android | 78 |
| Cadres de test d'applications iOS | 78 |

| | |
|---|-----|
| Cadres de test d'applications Web | 78 |
| Frameworks dans un environnement de test personnalisé | 78 |
| Support des versions Appium | 78 |
| Types de tests intégrés | 79 |
| Appium | 79 |
| Prise en charge des versions | 79 |
| Configurez votre package de test Appium | 80 |
| Création d'un fichier de package compressé | 91 |
| Téléchargez votre package de test sur Device Farm | 94 |
| Prenez des captures d'écran de vos tests (facultatif) | 95 |
| Tests sur Android | 96 |
| Cadres de test d'applications Android | 96 |
| Types de tests intégrés pour Android | 96 |
| Instrumentation | 96 |
| Tests iOS | 99 |
| Cadres de test d'applications iOS | 99 |
| Types de tests intégrés pour iOS | 99 |
| XCTest | 99 |
| XCTest UI | 102 |
| Tests d'applications Web | 104 |
| Règles relatives aux appareils mesurés et non mesurés | 104 |
| Tests intégrés | 104 |
| Types de tests intégrés | 105 |
| Intégré : Fuzz (Android et iOS) | 105 |
| Utilisation d'environnements de test personnalisés | 107 |
| Syntaxe des spécifications de test | 108 |
| Exemple de spécification de test | 110 |
| Environnement de test Android | 116 |
| Logiciels pris en charge | 117 |
| devicefarm-cli | 118 |
| Sélection de l'hôte de test Android | 119 |
| Exemple de fichier de spécifications de test | 120 |
| Migration vers l'hôte de test Amazon Linux 2 | 124 |
| Variables d'environnement | 127 |
| Variables d'environnement communes | 127 |
| Variables d'environnement Appium Java JUnit | 129 |

| | |
|---|-----|
| Variables d'environnement Appium Java TestNG | 129 |
| Variables d'environnement XcuiTest | 130 |
| Migration des tests | 130 |
| Considérations relatives à la migration | 130 |
| Étapes de la migration | 132 |
| Cadre Appium | 133 |
| Instrumentation Android | 133 |
| Migration des tests iOS XcuiTest existants | 133 |
| Extension du mode personnalisé | 133 |
| Configuration d'un code PIN | 133 |
| Accélérer les tests basés sur Appium grâce aux fonctionnalités souhaitées | 134 |
| Utilisation de webhooks et d'autres API après l'exécution de vos tests | 137 |
| Ajouter des fichiers supplémentaires à votre package de test | 138 |
| Utilisation de l'accès à distance | 142 |
| Création d'une session | 142 |
| Prérequis | 143 |
| Création d'une session avec la console Device Farm | 143 |
| Étapes suivantes | 144 |
| Utiliser une session | 144 |
| Prérequis | 144 |
| Utiliser une session dans la console Device Farm | 144 |
| Étapes suivantes | 145 |
| Trucs et astuces | 145 |
| Obtenir les résultats des sessions | 146 |
| Prérequis | 146 |
| Afficher les détails de la session | 146 |
| Téléchargement de vidéos ou de journaux de session | 146 |
| Utilisation d'appareils privés | 147 |
| Gestion des appareils privés | 148 |
| Création d'un profil d'instance | 148 |
| Gérer une instance d'appareil privée | 150 |
| Création d'un test ou d'une session d'accès à distance | 152 |
| Étapes suivantes | 153 |
| Sélection d'appareils privés | 153 |
| Règles ARN de l'appareil | 154 |
| Règles relatives aux étiquettes des instances de l'appareil | 155 |

| | |
|---|-----|
| Règles ARN de l'instance | 156 |
| Création d'un pool d'appareils privé | 156 |
| Création d'un pool d'appareils privés avec des appareils privés (AWS CLI) | 159 |
| Création d'un pool d'appareils privés avec des appareils privés (API) | 159 |
| Ignorer la nouvelle signature d'une application | 159 |
| Ignorer la nouvelle signature des applications sur les appareils Android | 161 |
| Ignorer la nouvelle signature d'une application sur les appareils iOS | 161 |
| Créer une session d'accès à distance pour faire confiance à votre application | 162 |
| Utilisation des services de point de terminaison VPC | 163 |
| Avant de commencer | 164 |
| Étape 1 : Création d'un équilibreur de charge réseau | 165 |
| Étape 2 : créer un service de point de terminaison VPC | 168 |
| Étape 3 : créer une configuration de point de terminaison VPC | 169 |
| Étape 4 : Création d'un essai | 170 |
| Travailler dans toutes les régions | 170 |
| Vue d'ensemble du peering VPC | 171 |
| Prérequis | 172 |
| Étape 1 : établir une connexion de peering entre deux VPC | 173 |
| Étape 2 : mise à jour des tables de routage pour VPC-1 et VPC-2 | 173 |
| Étape 3 : Création de groupes cibles | 174 |
| Étape 4 : Création d'un équilibreur de charge réseau | 176 |
| Étape 5 : créer un service de point de terminaison VPC | 177 |
| Étape 6 : Création d'une configuration de point de terminaison VPC dans l'application | 177 |
| Étape 7 : Création d'un essai | 178 |
| Création de systèmes VPC évolutifs | 178 |
| Résiliation d'appareils privés | 178 |
| Connectivité VPC | 179 |
| AWScontrôle d'accès et IAM | 181 |
| Rôles liés à un service | 182 |
| Autorisations de rôle liées à un service pour Device Farm | 183 |
| Création d'un rôle lié à un service pour Device Farm | 186 |
| Modification d'un rôle lié à un service pour Device Farm | 186 |
| Supprimer un rôle lié à un service pour Device Farm | 186 |
| Régions prises en charge pour les rôles liés au service Device Farm | 187 |
| Prérequis | 188 |
| Connexion à Amazon VPC | 189 |

| | |
|--|-----|
| Limites | 191 |
| Journalisation des appels d'API avec AWS CloudTrail | 192 |
| Informations sur AWS Device Farm dans CloudTrail | 192 |
| Comprendre les entrées du fichier journal d'AWS Device Farm | 193 |
| CodePipelineIntégration | 196 |
| Configurez CodePipeline pour utiliser vos tests Device Farm | 197 |
| Référence AWS CLI | 201 |
| Fenêtres PowerShell référence | 202 |
| Automatiser un parc d'appareils | 203 |
| Exemple : utilisation du AWS SDK pour démarrer une ferme de périphériques, exécuter et collecter des artefacts | 203 |
| Résolution des problèmes | 208 |
| Applications pour Android | 208 |
| ANDROID_APP_UNZIP_FAILED | 208 |
| ANDROID_APP_AAPT_DEBUG_BADGING_FAILED | 209 |
| ANDROID_APP_PACKAGE_NAME_VALUE_MISSING | 211 |
| ANDROID_APP_SDK_VERSION_VALUE_MISSING | 211 |
| ANDROID_APP_AAPT_DUMP_XMLTREE_FAILED | 212 |
| ANDROID_APP_DEVICE_ADMIN_PERMISSIONS | 213 |
| Certaines fenêtres de mon application Android affichent un écran vide ou noir | 215 |
| Appium Java JUnit | 215 |
| APPIUM_JAVA_JUNIT_TEST_PACKAGE_PACKAGE_UNZIP_FAILED | 216 |
| APPIUM_JAVA_JUNIT_TEST_PACKAGE_DEPENDENCY_DIR_MISSING | 217 |
| APPIUM_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DIR | 218 |
| APPIUM_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING | 219 |
| APPIUM_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JAR | 220 |
| APPIUM_JAVA_JUNIT_TEST_PACKAGE_JUNIT_VERSION_VALUE_UNKNOWN | 222 |
| APPIUM_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION | 223 |
| Appium Java JUnit Web | 224 |
| APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED | 224 |
| APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_DEPENDENCY_DIR_MISSING | 225 |
| APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DIR .. | 226 |
| APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING | 227 |
| APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JAR .. | 228 |
| APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_JUNIT_VERSION_VALUE_UNKNOWN ... | 230 |
| APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION | 231 |

| | |
|---|-----|
| Appium Java TestNG | 232 |
| APPIUM_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED | 232 |
| APPIUM_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSING | 233 |
| APPIUM_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DIR | 234 |
| APPIUM_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING | 235 |
| APPIUM_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JAR | 236 |
| Appium Java TestNG en ligne | 238 |
| APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED | 238 |
| APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSING | 239 |
| APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DIR | 240 |
| APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING | 241 |
| APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JAR | 242 |
| Appium Python | 244 |
| APPIUM_PYTHON_TEST_PACKAGE_UNZIP_FAILED | 244 |
| APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING | 245 |
| APPIUM_PYTHON_TEST_PACKAGE_INVALID_PLATFORM | 246 |
| APPIUM_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING | 247 |
| APPIUM_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME | 248 |
| APPIUM_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISSING | 250 |
| APPIUM_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION | 251 |
| APPIUM_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS_FAILED | 252 |
| APPIUM_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED | 253 |
| APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEELS_INSUFFICIENT | 255 |
| Site Web Appium Python | 256 |
| APPIUM_WEB_PYTHON_TEST_PACKAGE_UNZIP_FAILED | 256 |
| APPIUM_WEB_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING | 257 |
| APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PLATFORM | 258 |
| APPIUM_WEB_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING | 259 |
| APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME | 260 |
| APPIUM_WEB_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISSING | 261 |
| APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION | 262 |
| APPIUM_WEB_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS_FAILED | 264 |
| APPIUM_WEB_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED | 265 |
| Instrumentation | 266 |
| INSTRUMENTATION_TEST_PACKAGE_UNZIP_FAILED | 266 |
| INSTRUMENTATION_TEST_PACKAGE_AAPT_DEBUG_BADGING_FAILED | 267 |

| | |
|---|-----|
| INSTRUMENTATION_TEST_PACKAGE_INSTRUMENTATION_RUNNER_VALUE_MISSING | 269 |
| INSTRUMENTATION_TEST_PACKAGE_AAPT_DUMP_XMLTREE_FAILED | 270 |
| INSTRUMENTATION_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSING | 271 |
| Applications iOS | 272 |
| IOS_APP_UNZIP_FAILED | 273 |
| IOS_APP_PAYLOAD_DIR_MISSING | 273 |
| IOS_APP_APP_DIR_MISSING | 274 |
| IOS_APP_PLIST_FILE_MISSING | 275 |
| IOS_APP_CPU_ARCHITECTURE_VALUE_MISSING | 276 |
| IOS_APP_PLATFORM_VALUE_MISSING | 277 |
| IOS_APP_WRONG_PLATFORM_DEVICE_VALUE | 279 |
| IOS_APP_FORM_FACTOR_VALUE_MISSING | 280 |
| IOS_APP_PACKAGE_NAME_VALUE_MISSING | 282 |
| IOS_APP_EXECUTABLE_VALUE_MISSING | 283 |
| XCTest | 284 |
| XCTEST_TEST_PACKAGE_UNZIP_FAILED | 285 |
| XCTEST_TEST_PACKAGE_XCTEST_DIR_MISSING | 285 |
| XCTEST_TEST_PACKAGE_PLIST_FILE_MISSING | 286 |
| XCTEST_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING | 287 |
| XCTEST_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING | 288 |
| XCTest UI | 290 |
| XCTEST_UI_TEST_PACKAGE_UNZIP_FAILED | 290 |
| XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_MISSING | 291 |
| XCTEST_UI_TEST_PACKAGE_APP_DIR_MISSING | 292 |
| XCTEST_UI_TEST_PACKAGE_PLUGINS_DIR_MISSING | 293 |
| XCTEST_UI_TEST_PACKAGE_XCTEST_DIR_MISSING_IN_PLUGINS_DIR | 294 |
| XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING | 295 |
| XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING_IN_XCTEST_DIR | 296 |
| XCTEST_UI_TEST_PACKAGE_CPU_ARCHITECTURE_VALUE_MISSING | 297 |
| XCTEST_UI_TEST_PACKAGE_PLATFORM_VALUE_MISSING | 298 |
| XCTEST_UI_TEST_PACKAGE_WRONG_PLATFORM_DEVICE_VALUE | 300 |
| XCTEST_UI_TEST_PACKAGE_FORM_FACTOR_VALUE_MISSING | 301 |
| XCTEST_UI_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING | 303 |
| XCTEST_UI_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING | 304 |
| XCTEST_UI_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSING | 306 |
| XCTEST_UI_TEST_PACKAGE_TEST_EXECUTABLE_VALUE_MISSING | 307 |

| | |
|--|-----|
| Sécurité | 309 |
| Gestion des identités et des accès | 310 |
| Public ciblé | 310 |
| Authentification par des identités | 310 |
| Comment AWS Device Farm fonctionne avec IAM | 314 |
| Gestion des accès à l'aide de politiques | 319 |
| Exemples de politiques basées sur l'identité | 321 |
| Résolution des problèmes | 327 |
| Validation de la conformité | 330 |
| Protection des données | 330 |
| Chiffrement en transit | 332 |
| Chiffrement au repos | 332 |
| Conservation des données | 332 |
| Gestion des données | 333 |
| Gestion des clés | 334 |
| Confidentialité du trafic inter-réseau | 334 |
| Résilience | 334 |
| Sécurité de l'infrastructure | 335 |
| Sécurité de l'infrastructure pour les tests de périphériques physiques | 335 |
| Sécurité de l'infrastructure pour les tests de navigateurs de bureau | 336 |
| Analyse de la configuration et des vulnérabilités | 336 |
| Réponse aux incidents | 337 |
| Journalisation et surveillance | 337 |
| Bonnes pratiques de sécurité | 338 |
| Limites | 339 |
| Outils et plug-ins | 340 |
| Plug-in Jenkins CI | 340 |
| Étape 1 : Installer le plug-in | 343 |
| Étape 2 : créer un utilisateur IAM | 344 |
| Étape 3 : Instructions pour la toute première configuration | 345 |
| Étape 4 : Utiliser le plugin | 346 |
| Dépendances | 347 |
| Plug-in Device Farm Gradle | 347 |
| Création du plugin Device Farm Gradle | 347 |
| Configuration du plugin Device Farm Gradle | 348 |
| Génération d'un utilisateur IAM | 351 |

| | |
|--|---------|
| Configuration des types de tests | 352 |
| Dépendances | 354 |
| Historique du document | 355 |
| Glossaire AWS | 361 |
| | ccclxii |

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).

Device Farm peut être utilisé de deux manières principales :

- Tests automatisés d'applications à l'aide de divers frameworks de test.
- Accès à distance aux appareils sur lesquels vous pouvez charger des applications, les exécuter et interagir avec elles en temps réel.

Note

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

Tests automatisés des applications

Device Farm vous permet de télécharger vos propres tests ou d'utiliser des tests de compatibilité intégrés sans script. Étant donné que les tests sont exécutés en parallèle, les tests sur plusieurs appareils commencent en quelques minutes.

Au fur et à mesure que les tests sont terminés, un rapport de test contenant des résultats de haut niveau, des journaux de bas niveau, pixel-to-pixel captures d'écran et les données de performance sont mises à jour.

Device Farm permet de tester des applications Android et iOS natives et hybrides, y compris celles créées avec PhoneGap, Titanium, Xamarin, Unity et d'autres frameworks. Il prend en charge l'accès à distance à des applications Android et iOS à des fins de tests interactifs. Pour plus d'informations sur les types de tests pris en charge, consultez [Utilisation des types de tests dans AWS Device Farm](#).

Interaction d'accès à distance

L'accès à distance vous permet de faire glisser, d'appuyer et d'interagir avec un appareil par le biais de votre navigateur Web en temps réel. Il existe un certain nombre de situations où l'interaction en temps réel à l'aide d'un appareil est utile. Par exemple, les représentants du service client peuvent

aider les clients à utiliser ou à configurer leur appareil. Ils peuvent également montrer aux clients comment utiliser les applications s'exécutant sur un appareil spécifique. Vous pouvez installer des applications sur un appareil s'exécutant dans une session d'accès à distance, puis reproduire les problèmes des clients ou les problèmes détectés.

Au cours d'une session d'accès à distance, Device Farm collecte des informations sur les 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.

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

Une demande demandant à Device Farm de 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, voir [Tarification 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 plus d'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 plus d'informations, veuillez consulter [Exécutions](#).

séance

Interaction en temps réel avec un appareil physique réel via votre navigateur web. Pour plus d'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 <https://portal.aws.amazon.com/billing/signup>.
2. Suivez les instructions en ligne.

Dans le cadre de la procédure d'inscription, vous recevrez un appel téléphonique 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 root 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 administratif](#), et à uniquement utiliser l'utilisateur root pour effectuer [tâches nécessitant un accès utilisateur root](#).

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 AWS Identity and Access Management (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*.

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

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. veuillez consulter [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 us-west-2 (Oregon) AWS Région.

Rubriques

- [Prérequis](#)
- [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](#)

Prérequis

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

- Suivez les étapes de [Configuration](#). Vous avez besoin d'un AWS compte et un AWS Identity and Access Management utilisateur (IAM) autorisé à accéder à Device Farm.
- Pour Android, vous avez besoin d'un fichier .apk (package d'application Android). 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 ultérieurement au cours de 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 plus d'informations, veuillez consulter [Utilisation des types de tests 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 Tests d'appareils mobiles, puis choisissez Projets.
2. En dessous Projets de test d'appareils mobiles, choisissez Nouveau projet.
3. En dessous Création d'un projet, entrez un Nom du projet (par exemple, **MyDemoProject**).
4. Sélectionnez Create (Créer).


La console ouvre Tests automatisés page de votre 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 plus d'informations, veuillez consulter [Exécutions](#).

1. Sur la page Automated tests (Tests automatisés), choisissez Create a new run (Créer une nouvelle exécution).
2. Sur le Choisissez une application page, sous Appli mobile, choisissez Choisissez un fichier, puis choisissez un fichier Android (.apk) ou iOS (.ipa) sur votre ordinateur. Vous pouvez également faire glisser le fichier depuis votre ordinateur et le déposer dans la console.

3. Entrez unNom de la course, tels **quemy first test**. Par défaut, la console Device Farm utilise le nom du fichier.
4. Choisissez Suivant.
5. Sur leConfigurezpage, sousConfigurez le framework de test, choisissez l'un des frameworks de test ou des suites de tests intégrées. Pour plus d'informations sur chaque option, veuillez consulter [Types de tests et cadres](#).
 - Si vous n'avez pas encore empaqueté vos tests pour Device Farm, choisissezIntégré : Fuzzpour exécuter une suite de tests standard intégrée. Vous pouvez conserver les valeurs par défaut pourNombre d'événements,Accélérateur événementiel, etGraine du Randomizer. Pour plus d'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. Choisissez Suivant.
7. Sur leSélectionnez les appareilspage, pourPool d'appareils, choisissezPrincipaux appareils.
8. Choisissez Suivant.
9. Sur la page Specify device state (Spécifier l'état de l'appareil), effectuez l'une des opérations suivantes :
 - Pour fournir des données supplémentaires à utiliser par Device Farm pendant l'exécution, sousAjoutez des données supplémentaires, chargez un fichier .zip.
 - Pour installer d'autres applications pour l'exécution, sousInstaller d'autres applications, téléchargez les fichiers .apk ou .ipa pour les applications. Pour modifier l'ordre d'installation, glissez et déposez les fichiers.
 - Pour activer les radios Wi-Fi, Bluetooth, GPS ou NFC pendant la course, sousRégler les états de la radio, cochez les cases correspondantes.

 Note

Le réglage de l'état radio de l'appareil n'est actuellement disponible que pour les tests natifs d'Android.

- Pour tester le comportement spécifique à un emplacement pendant l'exécution, sousEmplacement de l'appareil, spécifiez le préréglageLatitudeetLongitudecoordonnées.

- Pour prédéfinir la langue et la région de l'appareil pour l'exécution, sousAppareil local, choisissez un lieu.
- Pour prédéfinir le profil réseau pour l'exécution, sousProfil du réseau, choisissez un profil sélectionné. Ou choisissezCréation d'un profil réseaupour créer le vôtre.

10. Choisissez Suivant.

11. Sur la page Review and start run (Vérifier et démarrer l'exécution), choisissez Confirm and start run (Confirmer et démarrer l'exécution).

Device Farm démarre l'exécution dès que les appareils sont disponibles, généralement en quelques minutes. Pour consulter l'état de l'exécution, surTests automatisépage de votre projet, choisissez le nom de votre course. Sur la page d'exécution, sousAppareils, chaque appareil commence par l'icône en

attente 

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

cours 

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 consulter les résultats des tests effectués en cours d'exécution, surTests automatisépage de votre projet, choisissez le nom de votre course. 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 plus d'informations, veuillez consulter [Utilisation des rapports de test dans Device Farm](#).

Étapes suivantes

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

Achetez 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.

Rubriques

- [Acheter des emplacements pour appareils \(console\)](#)
- [Achetez un emplacement pour appareil \(AWS CLI\)](#)
- [Acheter un emplacement pour appareil \(API\)](#)

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 Tests d'appareils mobiles, puis choisissez Fentes pour appareils.
3. Sur le [Achetez et gérez des emplacements pour appareils](#) page, vous pouvez créer votre propre package personnalisé en choisissant le nombre d'emplacements de Tests automatisés et Accès à distance appareils 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, voir [Tarification d'AWS Device Farm](#).

Important

Si vous modifiez le nombre d'emplacements de l'appareil mais que vous voyez un message, contactez-nous pour acheter un message, votre AWS le compte n'est pas encore approuvé pour acheter le nombre d'emplacements que vous avez demandés. Ces options vous invitent à envoyer un e-mail à l'équipe d'assistance de Device Farm. Dans l'e-mail, spécifiez 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

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

Next billing period

From August 16, you will have

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

Remote access

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

Current billing period

You currently have

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

Next billing period

From August 16, you will have

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

4. Choisissez Purchase (Acheter). UNConfirmer l'achat une fenêtre apparaît. Passez en revue les informations, puis choisissez Confirmer pour terminer la transaction.

Confirm purchase



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

Cancel

Confirm

Sur le [Achetez et gérez des emplacements pour appareil](#) sur cette page, vous pouvez voir le nombre d'emplacements d'appareils 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 gratuit restantes, exécutez `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 `lelist-offering-promotionscommande`.

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
  }
}
```

Les commandes `renew-offering`, `list-offering-transactions` et `des` sont également disponibles pour cette fonctionnalité. Pour plus d'informations, consultez le [Référence AWS CLI](#).

Acheter un emplacement pour appareil (API)

1. Appelez le [GetAccountSettings](#) opération pour répertorier les paramètres de votre compte.
2. Appelez le [ListOfferings](#) opération permettant de répertorier les offres d'emplacements pour appareils mises à votre disposition.
3. Appelez le [ListOfferingPromotions](#) opération permettant de 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 le [PurchaseOffering](#) opération d'achat d'une offre.
5. Appelez le [GetOfferingStatus](#) opération pour obtenir le statut de l'offre.

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

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

Concepts d'AWS Device Farm

Cette section décrit les concepts importants de Device Farm.

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

Pour plus d'informations sur les types de tests pris en charge dans Device Farm, consultez [Utilisation des types de tests 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 voir la liste complète des appareils, consultez [Liste des appareils](#).

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 [Utilisation d'appareils privés dans AWS Device Farm](#).

Identification de l'appareil

Device Farm effectue des tests sur des appareils mobiles et tablettes physiques de divers OEM.

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 disposent d'une connexion Wi-Fi 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 sont installés sur les appareils qui les prennent en charge, mais ces appareils ne disposent pas de compte Google actif.

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 plus d'informations, consultez [Utilisation d'environnements de test personnalisés](#).

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](#)
- [Applications d'instrumentation](#)
- [Resignature d'applications en plusieurs fois](#)
- [Applications obfusquées en cours 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. Ceux-ci incluent les coordonnées de latitude et de longitude, les paramètres régionaux, les états de radio (par exemple, Bluetooth, GPS, NFC et Wi-Fi), des données supplémentaires (contenues dans un fichier .zip), et des applications auxiliaires (qui doivent être installées avant que l'application soit testée).

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 plus d'informations, consultez [Définissez le délai d'exécution pour les tests dans AWS Device Farm](#).

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 d'UUID Device Farm 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.

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 [Utilisation des tests dans AWS Device Farm](#).

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 plus d'informations, consultez [Utilisation des rapports de test dans Device Farm](#).

Sessions dans AWS Device Farm

Vous pouvez utiliser Device Farm pour effectuer des tests interactifs d'applications Android et iOS par le biais de sessions d'accès à distance dans un navigateur Web. Ce type de test interactif permet, lors d'un appel client, d'aider les ingénieurs à résoudre le problème du client, étape par étape. Les développeurs peuvent reproduire un problème sur un appareil spécifique pour isoler l'origine du problème. Vous pouvez utiliser des sessions à distance pour effectuer des tests d'utilisation avec vos clients cibles.

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 plus d'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 resigne 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 à partir de produits tels que DexGuard pour 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 DexGuard avec les mesures de lutte contre le piratage, Device Farm ne peut pas signer à nouveau l'application.

Travailler sur des 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, l'interface en ligne de commande (AWS CLI), ou l'API AWS Device Farm pour travailler avec 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, l'interface en ligne de commande (AWS CLI), ou l'API AWS Device Farm.

Prérequis

- 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 Tests d'appareils mobiles, puis choisissez Projets.
3. Choisissez New Project (Nouveau projet).
4. Entrez un nom pour votre projet, puis choisissez Soumettre.
5. Pour spécifier les paramètres pour le projet, choisissez Project settings (Paramètres du projet). Ces paramètres incluent le délai d'attente par défaut pour les exécutions de test. Une fois que les paramètres sont appliqués, ils sont utilisés dans toutes les exécutions de test du projet. Pour

plus d'informations, veuillez consulter [Définissez le délai d'exécution pour les tests dans AWS Device Farm](#).

Créez un projet (AWS CLI)

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

Exemple :

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

La réponse de l'AWS CLI 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 [Référence AWS CLI](#).

Création d'un projet (API)

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

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

Afficher la liste des projets dans AWS Device Farm

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

Rubriques

- [Prérequis](#)

- [Afficher la liste des projets \(console\)](#)
- [Consultez 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, sélectionnez Tests d'appareils mobiles, puis choisissez Projets.
 - Pour les projets de test de navigateurs de bureau, dans le menu de navigation de Device Farm, sélectionnez Test du navigateur de bureau, puis choisissez Projets.

Consultez la liste des projets (AWS CLI)

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

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

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

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 un parc d'appareils](#).

Utilisation des tests 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 AWS Command Line Interface (AWS CLI) ou l'API AWS Device Farm pour travailler avec des runs.

Rubriques

- [Création d'un test dans Device Farm](#)
- [Définissez le délai d'exécution pour les tests dans AWS Device Farm](#)
- [Simulez les connexions réseau et les conditions d'exécution 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 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

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

Prérequis

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](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 a new run (Créer une exécution).
5. Sur la page Choisir une application, choisissez Application mobile ou Application Web.

Step 1
Choose application

Step 2
Configure

Step 3
Select devices

Step 4
Specify device state

Step 5
Review and start run

Choose application

Mobile App | Web App

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

Choose File or drop file here or Select a recent upload

Cancel Next step

6. Chargez votre fichier d'application. Vous pouvez également faire glisser et déposer votre fichier ou choisir un chargement récent. Si vous chargez une application iOS, veillez à choisir iOS device (Appareil iOS) et non un simulateur.
7. (Facultatif) Dans Run name (Nom de l'exécution), entrez un nom. Par défaut, Device Farm utilise le nom du fichier de l'application.
8. Choisissez Suivant.
9. Sur la page Configure (Configurer), choisissez l'une des suites de tests 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 [Utilisation des types de tests dans AWS Device Farm](#).

10. Si vous n'avez pas choisi Built-in : Fuzz, sélectionnez Choisir un fichier, puis 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](#).
12. Si vous utilisez l'environnement de test standard, passez à l'étape 13. Si vous utilisez un environnement de test personnalisé avec le fichier YAML de spécification de test par défaut, passez à l'étape 13.
 - a. 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.
 - b. Si vous avez modifié la spécification de test, choisissez Enregistrer en tant que nouveau pour la mettre à jour.
13. Si vous souhaitez configurer les options d'enregistrement vidéo ou de capture des données de performance, choisissez Advanced Configuration (Configuration avancée).
 - a. Sélectionnez Activer l'enregistrement vidéo pour enregistrer une vidéo pendant le test.
 - b. Sélectionnez Activer la capture des données de performance des applications pour capturer les données de performance de l'appareil.

 Note

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

14. Choisissez Suivant.
15. Sur la page Select devices (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 [Créer un groupe 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.

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

16. Choisissez Suivant.

17. Sur la page Specify device state (Spécifier l'état de l'appareil) :

- 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.

18. Choisissez Suivant.

19. Sur la page Vérifier et démarrer l'exécution, vous pouvez spécifier le délai d'exécution de votre test. Si vous utilisez des emplacements de test illimités, vérifiez que l'option Run on unmetered slots (Exécuter sur des emplacements illimités) est sélectionnée.

20. Entrez une valeur ou utilisez la barre du curseur pour modifier le délai d'exécution. Pour de plus amples informations, veuillez consulter [Définissez le délai d'exécution pour les tests dans AWS Device Farm](#).

21. Choisissez Confirm and start run (Confirmer et démarrer 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 d'URL présignées 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](#).

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
```

- 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.
- 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"
  }
}
```

- Notez l'ARN de chargement de la spécification de test et l'URL présignée.
- 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"
```

- 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 [Référence AWS CLI](#).

Création d'un test (API)

Les étapes sont les mêmes que celles décrites dans la AWS CLI section. veuillez consulter [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. veuillez consulter [CreateUpload](#).

- L'ARN de chargement d'un package de test. veuillez consulter [CreateUpload](#).
- L'ARN d'un groupe d'appareils. Consultez [Créer un groupe 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 un parc d'appareils](#).

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 [Utilisation des rapports de test dans Device Farm](#).

Définissez le 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.

Prérequis

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](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](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).

Simulez les connexions réseau et les conditions d'exécution 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, FireOS et Web dans Device Farm. Par exemple, vous pouvez tester votre application dans des conditions réseau qui ne sont pas parfaites.

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 plus d'informations, consultez [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](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: [redacted] Stop run

No recent tests

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

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

0 out of 5 devices completed 0%

Devices | Unique problems | Screenshots | Parsing result

Devices

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

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

5. 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.

| 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 l'exécution de test spécifiée, où *myARN* correspond à l'ARN (Amazon Resource Name) de l'exécution de 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 [Référence AWS CLI](#).

Arrêter une exécution (API)

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

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

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](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 [Référence AWS CLI](#).

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 un parc d'appareils](#).

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

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

Prérequis

- 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 Sélectionner les appareils, choisissez Créer un pool d'appareils.
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 valeur de type, choisissez Form Factor.
- d. Pour Opérateur (deuxième liste déroulante), choisissez EQUALS pour inclure les appareils dont la valeur du champ est égale à la valeur de la valeur.
- e. Pour Valeur (troisième liste déroulante), entrez ou choisissez la valeur que vous souhaitez spécifier pour les valeurs de champ et d'opérateur. Si vous choisissez Platform (Plateforme) pour Field (Champ), 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.
- g. Pour supprimer une règle, choisissez l'icône X en regard de la 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.

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.

 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, veillez 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)

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

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

Création d'un pool d'appareils (API)

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

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

Analyse des résultats 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.

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.

Rubriques

- [Utilisation des rapports de test dans Device Farm](#)
- [Utilisation d'artefacts dans Device Farm](#)

Utilisation des 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

- [Prérequis](#)
- [Comprendre les résultats des tests](#)
- [Affichage des rapports](#)

Prérequis

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 démarrera 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 [Comprendre les résultats des tests](#).

icône

Comprendre les résultats des tests

La console Device Farm affiche des icônes qui vous aident à évaluer rapidement l'état de votre cycle de test terminé.

Rubriques

- [Communication des résultats d'un test individuel](#)
- [Communication des résultats de plusieurs tests](#)

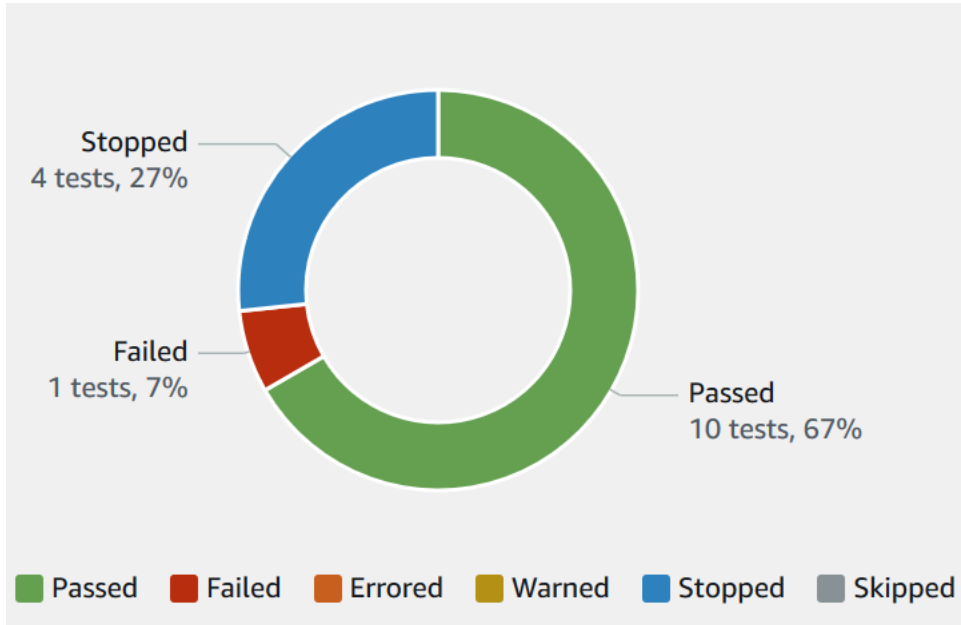
Communication des résultats d'un test individuel

Pour les rapports décrivant un test individuel, Device Farm affiche une icône :

| 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. | ⊖ |

Communication des résultats de plusieurs tests

Si vous choisissez une exécution terminée, Device Farm affiche un graphique récapitulatif des résultats des tests.



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.

Affichage des 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 données de performance relatives à un problème, à un appareil, à une suite ou à un test dans un rapport](#)
- [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](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 <input type="text" value="Find device by status, device name, or OS"/> < 1 > | | | | |
| Status | Device | OS | Test Results | Total Minutes |
| ✓ Passed | Google Pixel 4 XL (Unlocked) | 10 | Passed: 3, errored: 0, failed: 0 | 00:02:36 |
| ✓ Passed | Samsung Galaxy S20 (Unlocked) | 10 | Passed: 3, errored: 0, failed: 0 | 00:02:34 |
| ✗ Failed | Samsung Galaxy S20 ULTRA (Unlocked) | 10 | Passed: 2, errored: 0, failed: 1 | 00:02:25 |
| ✓ Passed | Samsung Galaxy S9 (Unlocked) | 9 | Passed: 3, errored: 0, failed: 0 | 00:02:46 |
| ✓ Passed | Samsung Galaxy Tab S4 | 8.1.0 | Passed: 3, errored: 0, failed: 0 | 00:03:13 |

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 [Communication des résultats 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 Performance affiche des informations sur les données de performance générées par Device Farm pendant le test. Pour afficher ces données de performance, suivez les instructions fournies dans [Afficher les données de performance 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 [Communication des résultats de 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 Performance affiche des informations sur les données de performance générées par Device Farm pour l'appareil pendant l'exécution. Pour afficher ces données de performance,

suivez les instructions fournies dans [Afficher les données de performance 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 [Communication des résultats de 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 Performance affiche des informations sur les données de performance générées par Device Farm lors de l'exécution de la suite. Pour afficher ces données de performance, suivez les instructions fournies dans [Afficher les données de performance 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 [Communication des résultats 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 Performance affiche des informations sur les données de performance générées par Device Farm pendant le test. Pour afficher ces données de performance, suivez les instructions fournies dans [Afficher les données de performance 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 données de performance relatives à un problème, à un appareil, à une suite ou à un test dans un rapport

Note

Device Farm collecte les données de performance des appareils uniquement pour les appareils Android pour le moment.

L'onglet Performances affiche les informations suivantes :

- Le graphique du processeur affiche le pourcentage de processeur utilisé par l'application sur un seul cœur lors du problème, de l'appareil, de la suite ou du test sélectionné (le long de l'axe vertical) au fil du temps (le long de l'axe horizontal).

L'axe vertical est exprimé en pourcentage, de 0 % jusqu'au pourcentage maximal enregistré.

Ce pourcentage peut dépasser 100 % si l'application a utilisé plusieurs cœurs. Par exemple, si trois cœurs sont à 60 % d'utilisation, le pourcentage affiché est 180 %.

- Le graphique de la mémoire indique le nombre de Mo utilisés par l'application lors du problème, de l'appareil, de la suite ou du test sélectionné (le long de l'axe vertical) au fil du temps (le long de l'axe horizontal).

L'axe vertical est exprimé en Mo, de 0 Mo jusqu'au nombre maximal de Mo enregistrés.

- Le graphique Threads affiche le nombre de threads utilisés pendant le problème, l'appareil, la suite ou le test sélectionné (axe vertical) au fil du temps (axe horizontal).

L'axe vertical est exprimé en nombre de fils, de zéro fil au nombre maximum de fils enregistrés.

Dans tous les cas, l'axe horizontal est représenté, en secondes, du début à la fin de l'exécution du problème, appareil, suite ou test sélectionné.

Pour afficher des informations concernant un point de données spécifique, suspendez le graphique souhaité à la seconde souhaitée sur l'axe horizontal.

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 au format *MM:SS.SSS*, où *M* représente les minutes et *S* représente les 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)**

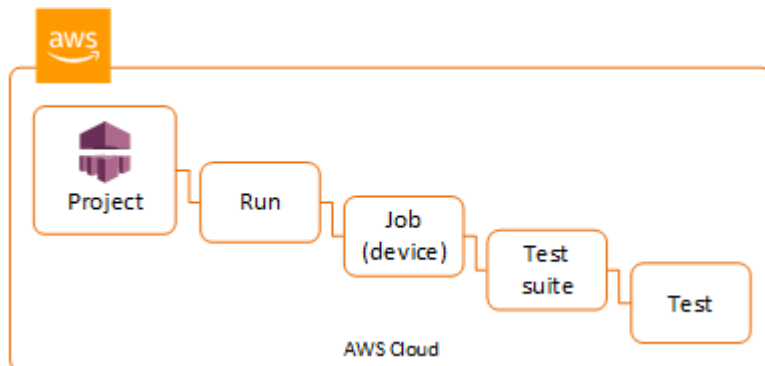
- Tag (Balise) représente les métadonnées arbitraires de l'entrée de journal. Par exemple, Android logcat peut s'en servir pour décrire la partie du système ayant créé l'entrée de journal (par exemple, `ActivityManager`).
- Message représente le message ou les données de l'entrée de journal. Par exemple, `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.

Utilisation 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 :

Dépôt de

Fichiers générés pendant le test, y compris les rapports Device Farm. Pour de plus amples informations, veuillez consulter [Utilisation des 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.

Utilisation d'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 [Utilisation d'artefacts dans un environnement de test personnalisé](#).

Utilisation d'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 : Listez 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 : Listez 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

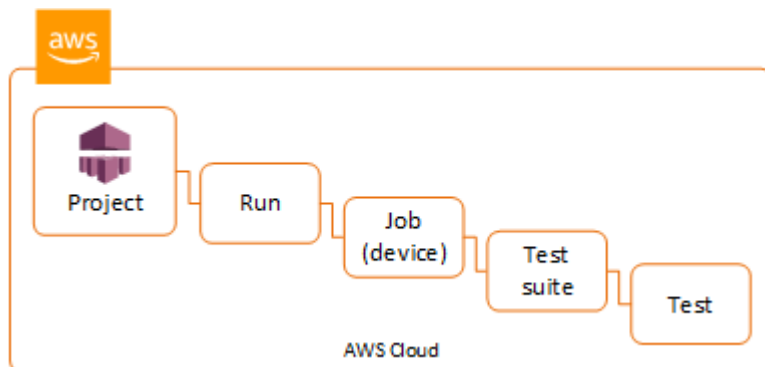
- Téléchargez votre artefact à l'aide de l'URL de l'étape précédente. Cet exemple utilise curl pour télécharger un fichier de sortie Appium Java Android :

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

Utilisation d'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.

Utilisation 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 [Utilisation d'artefacts dans Device Farm](#).

Balisage des ressources AWS Device Farm

AWS Device Farm fonctionne avec leAWSAPI de balisage des groupes de ressources. 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

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

Étiquetage des 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 d'ARN 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. Ainsi, dans l'exemple Python suivant, une série d'ARN de projet est balisée à l'aide de la balise `build-config` ayant 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`) avec le `tagstack` qui ont 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"])
```

Utilisation des types de tests dans AWS Device Farm

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

Cadres de test

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

Cadres de test d'applications Android

- [Travailler avec Appium et AWS Device Farm](#)
- [Utilisation d'instruments pour Android et AWS Device Farm](#)

Cadres de test d'applications iOS

- [Travailler avec Appium et AWS Device Farm](#)
- [Utilisation de XCTest pour iOS et AWS Device Farm](#)
- [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 [Travailler avec Appium et AWS 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 framework XCTest. Pour de plus amples informations, veuillez consulter [Utilisation d'environnements de test personnalisés](#).

Support des versions 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](#).

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 \(Android et iOS\)](#)

Travailler avec Appium et AWS Device Farm

Cette section décrit comment configurer, emballer et télécharger vos tests Appium sur 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.

Prise en charge des versions

La prise en charge de divers frameworks et langages de programmation dépend du langage utilisé.

Device Farm prend en charge toutes les versions de serveur Appium 1.x et 2.x. Pour Android, vous pouvez choisir n'importe quelle version majeure d'Appium avec `devicefarm-cli`. Par exemple, pour utiliser le serveur Appium version 2, ajoutez les commandes suivantes à votre fichier YAML de spécifications de test :

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

Pour iOS, vous pouvez choisir des versions spécifiques d'Appium à l'aide des commandes `avm` ou `ornpm`. Par exemple, pour utiliser la `avm` commande permettant de définir la version du serveur Appium sur 2.1.2, ajoutez les commandes suivantes à votre fichier YAML de spécifications de test :

```
phases:
```

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

À l'aide de la npm commande permettant d'utiliser la dernière version d'Appium 2, ajoutez les commandes suivantes à votre fichier YAML de spécifications de test :

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

Pour plus d'informations sur les commandes de l'interface de ligne de commande `devicefarm-cli` ou sur toute autre commande de l'interface de ligne de commande, consultez la [référence de l'interface de ligne de commande AWS](#).

Pour utiliser toutes les fonctionnalités du framework, telles que les annotations, choisissez un environnement de test personnalisé et utilisez la CLI AWS ou la Device Farm console pour télécharger une spécification de test personnalisée.

Rubriques

- [Configurez votre package de test Appium](#)
- [Création d'un fichier de package de test compressé](#)
- [Téléchargez votre package de test sur Device Farm](#)
- [Prenez des captures d'écran de vos tests \(facultatif\)](#)

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'emballage 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 empaqueter 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, telle que la **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](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 a new run (Créer une exécution).
5. Pour les tests natifs Android et iOS

Sur la page Choisir une application, choisissez Application mobile, puis sélectionnez Choisir un fichier 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

Sur la page Choisir une application, sélectionnez Web App.

6. Donnez à votre test un nom approprié. Il peut contenir n'importe quelle combinaison d'espaces ou de ponctuation.
7. Choisissez Suivant.
8. Sur la page Configurer, dans la section Configuration du cadre de test, choisissez la **Langue** Appium, puis Choisissez un fichier.
9. Recherchez et choisissez le fichier .zip contenant vos tests. Le fichier .zip doit respecter le format décrit dans [Configurez votre package de test Appium](#).
10. Choisissez Exécuter votre test dans un environnement personnalisé. Cet environnement d'exécution permet de contrôler totalement la configuration, le démontage et l'invocation des tests, ainsi que de choisir des versions spécifiques des environnements d'exécution et du serveur Appium. Vous pouvez configurer votre environnement personnalisé via le fichier de spécifications de test. Pour plus d'informations, consultez la section [Utilisation d'environnements de test personnalisés dans AWS Device Farm](#).
11. Choisissez Next, puis 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.

Utilisation des 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.

Cadres de test d'applications Android

Les tests suivants sont disponibles pour les appareils Android.

- [Travailler avec Appium et AWS Device Farm](#)
- [Utilisation d'instruments pour Android et AWS Device Farm](#)

Types de tests intégrés pour Android

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

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

Utilisation d'instruments pour Android et AWS Device Farm

Device Farm fournit un support pour l'instrumentation (JUnit, Espresso, 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.

Rubriques

- [Qu'est-ce que l'instrumentation ?](#)
- [Téléchargez vos tests d'instrumentation Android](#)
- [Prendre des captures d'écran lors de tests d'instrumentation Android](#)
- [Considérations supplémentaires pour les tests d'instrumentation Android](#)
- [Analyse syntaxique des tests en mode standard](#)

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.

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](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. Si le bouton Create a new run (Créer une exécution) s'affiche, choisissez-le.
5. Sur la page Choisir une application, sélectionnez Choisir un fichier.
6. Recherchez et sélectionnez votre fichier d'application Android. Le fichier doit être au format .apk.
7. Choisissez Suivant.
8. Sur la page Configurer, dans la section Configuration du cadre de test, choisissez Instrumentation, puis sélectionnez Choisir un fichier.
9. Recherchez et choisissez le fichier .apk contenant vos tests.
10. Choisissez Next, puis suivez les instructions restantes pour sélectionner les appareils et démarrer l'exécution.

Prendre des captures d'écran lors de 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`

Considérations supplémentaires pour les tests d'instrumentation 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 [android.support.test.Runner.AndroidJ UnitRunner](#).

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 de test et les méthodes 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
```

Utilisation des 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.

Cadres de test d'applications iOS

Les tests suivants sont disponibles pour les appareils iOS.

- [Travailler avec Appium et AWS Device Farm](#)
- [Utilisation de XCTest pour iOS et AWS Device Farm](#)
- [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 \(Android et iOS\)](#)

Utilisation de XCTest pour iOS et AWS Device Farm

Avec Device Farm, vous pouvez utiliser le framework XCTest pour tester votre application sur de vrais appareils. Pour plus d'informations sur XCTest, consultez [Testing Basics](#) dans Testing with Xcode.

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

Rubriques

- [Création des packages pour votre exécution de XCTest](#)
- [Téléchargement des packages pour votre exécution de XCTest sur Device Farm](#)

Création des packages pour votre exécution de XCTest

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

- Votre package d'application en tant que fichier `.ipa`.
- Votre package XCTest en tant que fichier `.zip`.

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 package XCTest

1. Dans l'outil de recherche, dans le répertoire `Debug-iphoneros`, 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.

Téléchargement des packages pour votre exécution de XCTest 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](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 a new run` (Créer une exécution).
5. Sur la page `Choisir une application`, sélectionnez `Application mobile`.
6. Sélectionnez `Choisir un fichier`.
7. Accédez au fichier `.ipa` pour votre application et chargez-le.

 Note

Votre package `.ipa` doit être conçu pour les tests.

8. Une fois le téléchargement terminé, choisissez Next.
9. Sur la page Configurer, dans la section Configuration du cadre de test, choisissez XCTest. Sélectionnez ensuite Choisir un fichier.
10. Accédez au fichier `.zip` qui contient le package XCTest pour votre application et chargez-le.
11. Une fois le téléchargement terminé, choisissez Next.
12. 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.
13. Après avoir configuré votre exécution, sur la page Vérifier et démarrer l'exécution, sélectionnez Confirmer et démarrer l'exécution.

Device Farm exécute votre test et affiche les résultats dans la console.

Utilisation du framework de test d'interface utilisateur XCTest pour iOS et AWS Device Farm

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

Rubriques

- [Qu'est-ce que le framework de test XCTest UI ?](#)
- [Préparez vos tests d'interface utilisateur iOS XCTest](#)
- [Téléchargez vos tests d'interface utilisateur iOS XCTest](#)
- [Prendre des captures d'écran lors des tests de l'interface utilisateur iOS XCTest](#)

Qu'est-ce que le framework de test XCTest UI ?

Le framework XCTest UI est le nouveau framework de test introduit avec Xcode 7. Ce framework étend XCTest avec des fonctionnalités de test d'interface utilisateur. Pour plus d'informations, consultez [User Interface Testing](#) dans la bibliothèque pour les développeurs iOS.

Préparez vos tests d'interface utilisateur iOS XCTest

Votre solution groupée d'exécuteur de test XCTest UI iOS doit être contenue dans un fichier .ipa au bon format.

Pour créer un fichier .ipa, placez votre bundle my-project-name UITest-Runner.app dans un répertoire de charge utile vide. Ensuite, archivez le répertoire Payload dans un fichier .zip, puis remplacez l'extension du fichier par .ipa. La solution groupée *UITest-Runner.app est générée par Xcode lorsque vous créez votre projet à des fins de test. Vous pouvez la trouver dans le répertoire Products de votre projet.

Téléchargez vos tests d'interface utilisateur iOS XCTest

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](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. Si le bouton Create a new run (Créer une exécution) s'affiche, choisissez-le.
5. Sur la page Choisir une application, sélectionnez Choisir un fichier.
6. 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.

7. Choisissez Suivant.
8. Sur la page Configurer, dans la section Configuration du cadre de test, choisissez XCTest UI, puis sélectionnez Choisir un fichier.

9. Recherchez et sélectionnez le fichier .ipa contenant votre exécuteur de test XCTest UI iOS.
10. Choisissez Next, puis suivez les instructions restantes pour sélectionner les appareils sur lesquels exécuter vos tests et démarrer l'exécution.

Prendre des captures d'écran lors des tests de l'interface utilisateur iOS XCTest

Les tests XCTest UI effectuent des captures d'écran automatiquement à chacune de leurs étapes. Ces captures d'écran sont affichées dans votre rapport de test de Device Farm. Aucun code supplémentaire n'est requis.

Utilisation des 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 "Appium"](#)

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 [Tarification 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.

Utilisation de tests intégrés dans AWS Device Farm

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

Types de tests intégrés

Les tests intégrés vous permettent de tester vos applications sans écrire de scripts.

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

Utilisation du test de fuzz intégré pour Device Farm

Device Farm propose un type de test de fuzz intégré.

Qu'est-ce que le test de fuzz intégré ?

Le test Fuzz intégré envoie de manière aléatoire des événements d'interface utilisateur aux appareils et fournit ensuite les résultats.

Utilisez le type de test de fuzz intégré

Utilisez la console 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](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. Si le bouton Create a new run (Créer une exécution) s'affiche, choisissez-le.
5. Sur la page Choisir une application, sélectionnez Choisir un fichier.
6. Recherchez et sélectionnez le fichier d'application dans lequel vous voulez exécuter le test Fuzz intégré.
7. Choisissez Suivant.
8. Sur la page Configurer, dans la section Configuration du cadre de test, choisissez Built-in : Fuzz.

9. 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.
10. Choisissez Next, puis suivez les instructions restantes pour sélectionner les appareils et démarrer l'exécution.

Utilisation d'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 évite les frais de prétraitement/post-traitement.
- 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

- [Syntaxe des spécifications de test](#)
- [Exemple de spécification de test](#)
- [Utilisation de l'environnement de test Amazon Linux 2 pour les tests Android](#)
- [Variables d'environnement](#)
- [Migration des tests d'un environnement de test standard vers un environnement de test personnalisé](#)
- [Extension des environnements de test personnalisés dans Device Farm](#)

Syntaxe des spécifications de test

Voici la structure du fichier de spécification de test YAML :

```
version: 0.1

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

artifacts:
  - location
  - location
```

La spécification de test contient les éléments suivants :

version

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

phases

Cette section contient les groupes de commandes exécutés au cours d'un test.

Les noms de phase de test autorisés sont :

install

Facultatif.

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

Facultatif.

Éventuelles commandes qui sont exécutées avant votre test automatisé.

test

Facultatif.

Commandes qui sont exécutées pendant votre test automatisé. Si une commande de la phase de test échoue, le test est marqué comme ayant échoué.

post_test

Facultatif.

Éventuelles commandes qui sont exécutées après votre test automatisé.

artifacts

Facultatif.

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 [Utilisation 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](#).

Exemple de spécification de test

Voici un exemple de spécification de test YAML de Device Farm qui configure un test Appium Java TestNG :

```
version: 0.1

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

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

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

  # For Android tests running on the Amazon Linux 2 test host, many software libraries
  # are available
  # from the test host using the devicefarm-cli tool. To learn more, please see:
```

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

# For iOS tests, you can use the Node.JS tools nvm, npm, and avm to setup your
environment. By
# default, Node.js versions 16.20.2 and 14.19.3 are available on the test host.
install:
  commands:
    # The Appium server is written using Node.js. In order to run your desired
    version of Appium,
    # you first need to set up a Node.js environment that is compatible with your
    version of Appium.
    - |-
      if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "Android" ];
      then
        devicefarm-cli use node 16;
      else
        # For iOS, use "nvm use" to switch between the two preinstalled NodeJS
versions 14 and 16,
        # and use "nvm install" to download a new version of your choice.
        nvm use 16;
      fi;
    - node --version

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

```
# Note that, for iOS devices, Appium 2 is only supported on iOS version 14
and above using
# NodeJS version 16 and above.
  avm 2.2.1;
fi;
- appium --version

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

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

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

# Install the NodeJS dependencies.
- cd $DEVICEFARM_TEST_PACKAGE_PATH
# First, install dependencies which were packaged with the test package using
npm-bundle.
- npm install *.tgz
# Then, optionally, install any additional dependencies using npm install.
```

```
# If you do run these commands, we strongly recommend that you include your
package-lock.json
# file with your test package so that the dependencies installed on Device Farm
match
# the dependencies you've installed locally.
# - cd node_modules/*
# - npm install

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

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

    # Appium downloads Chromedriver using a feature that is considered insecure for
    multitenant
    # environments. This is not a problem for Device Farm because each test host is
    allocated
```

```

# exclusively for one customer, then terminated entirely. For more information,
please see
# https://github.com/appium/appium/blob/master/packages/appium/docs/en/guides/
security.md

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

# This code will wait until the Appium server starts.

```

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

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

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

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

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



```
- $DEVICEFARM_LOG_DIR
```

Utilisation de l'environnement de test Amazon Linux 2 pour les tests 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 de test Amazon Linux 2 est le plus récent environnement de test Android, remplaçant l'ancien système basé sur Ubuntu. À l'aide de votre fichier de spécifications de test, vous pouvez choisir d'exécuter vos tests Android sur l'environnement Amazon Linux 2.

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` outil permet le développement de fichiers de spécifications de test avec une expérience cohérente dans tous les frameworks.

Rubriques

- [Logiciels pris en charge](#)
- [l'outil devicefarm-cli](#) ;
- [Sélection de l'hôte de test Android](#)
- [Exemple de fichier de spécifications de test](#)
- [Migration vers l'hôte de test Amazon Linux 2](#)

Logiciels pris en charge

L'hôte de test Amazon Linux 2 est préinstallé avec de nombreuses bibliothèques logicielles nécessaires pour prendre en charge les frameworks de test Device Farm, fournissant ainsi un environnement de test prêt au lancement. 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 plus d'informations, consultez l'[exemple de fichier de spécifications de test](#).

Les versions logicielles suivantes sont actuellement disponibles sur l'hôte :

| Software Library | Software Version | Command to use in your test spec file |
|------------------|------------------|---|
| Python | 3.8 | <code>devicefarm-cli utilise python 3.8</code> |
| | 3.9 | <code>devicefarm-cli utilise python 3.9</code> |
| | 3.10 | <code>devicefarm-cli utilise python 3.10</code> |
| Java | 8 | <code>devicefarm-cli utilise Java 8</code> |
| | 11 | <code>devicefarm-cli utilise Java 11</code> |
| | 17 | <code>devicefarm-cli utilise Java 17</code> |
| NodeJS | 16 | <code>devicefarm-cli utilise le nœud 16</code> |
| | 18 | <code>devicefarm-cli utilise le nœud 18</code> |

| | | |
|--------|-----|--|
| Ruby | 2.7 | <code>devicefarm-cli</code> <code>utilise ruby 2.7</code> |
| | 3.2 | <code>devicefarm-cli</code> <code>utilise ruby 3.2</code> |
| Appium | 1 | <code>devicefarm-cli</code> <code>utilise Appium 1</code> |
| | 2 | <code>devicefarm-cli</code> <code>utilise Appium 2</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 pilote Appium UIAutomator2) pour des outils tels qu'Appium. Cela garantit que vous disposez des outils nécessaires pour travailler avec les frameworks de test pris en charge.

l'outil **devicefarm-cli** ;

L'hôte de test Amazon Linux 2 utilise un outil de gestion de version standardisé appelé `devicefarm-cli` à sélectionner les versions du logiciel. Cet outil est distinct du Device Farm Test Host 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.

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 changer la version d'Appium de 1 à 2 :

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

Tip

Notez que lorsque vous sélectionnez une version logicielle, vous `devicefarm-cli` change également les outils de support pour ces langages, tels que `pip` Python et `npm` NodeJS.

Sélection de l'hôte de test Android

Pour les tests Android, Device Farm a besoin du champ suivant dans votre fichier de spécifications de test pour choisir l'hôte de test Amazon Linux 2 :

```
android_test_host: amazon_linux_2 | legacy
```

`amazon_linux_2` À utiliser pour exécuter vos tests sur l'hôte de test Amazon Linux 2 :

```
android_test_host: amazon_linux_2
```

Pour en savoir plus sur les avantages d'Amazon Linux 2, [cliquez ici](#).

Device Farm recommande d'utiliser l'hôte Amazon Linux 2 pour les tests Android plutôt que l'ancien environnement hôte. Si vous préférez utiliser l'ancien environnement, utilisez-le `legacy` pour exécuter vos tests sur l'ancien hôte de test :

```
android_test_host: legacy
```

Par défaut, les fichiers de spécifications de test sans sélection d'hôte de test s'exécuteront sur l'ancien hôte de test.

Syntaxe déconseillée

Vous trouverez ci-dessous la syntaxe obsolète permettant de choisir Amazon Linux 2 dans votre fichier de spécifications de test :

```
preview_features:  
  android_amazon_linux_2_host: true
```

Si vous utilisez cet indicateur, vos tests continueront de s'exécuter sur Amazon Linux 2. Cependant, nous vous recommandons vivement de supprimer la section `preview_features` des drapeaux et de la remplacer par le nouveau `android_test_host` champ afin d'éviter des frais de maintenance à l'avenir.

Warning

L'utilisation à la fois `android_amazon_linux_2_host` des indicateurs `android_test_host` et dans votre fichier de spécifications de test renverra une erreur. Un seul doit être utilisé ; nous le recommandons `android_test_host`.

Exemple de fichier de spécifications de test

L'extrait suivant est un exemple de fichier de spécifications de test Device Farm qui configure un test Appium NodeJS exécuté à l'aide de l'hôte de test Amazon Linux 2 pour Android :

```
version: 0.1  
  
# This flag enables your test to run using Device Farm's Amazon Linux 2 test host. For  
# more information,  
# please see https://docs.aws.amazon.com/devicefarm/latest/developerguide/amazon-  
linux-2.html  
android_test_host: amazon_linux_2  
  
# Phases represent collections of commands that are executed during your test run on  
# the test host.
```

phases:

```
# The install phase contains commands for installing dependencies to run your tests.
# For your convenience, certain dependencies are preinstalled on the test host. To
lean about which
# software is included with the host, and how to install additional software, please
see:
# https://docs.aws.amazon.com/devicefarm/latest/developerguide/amazon-linux-2-
supported-software.html
```

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

install:**commands:**

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

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

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

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

```
# - appium driver uninstall uiautomator2
# - appium driver install uiautomator2@2.34.0

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

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

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

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

    # We recommend starting the Appium server process in the background using the
command below.
    # The Appium server log will be written to the $DEVICEFARM_LOG_DIR directory.
    # The environment variables passed as capabilities to the server will be
automatically assigned
    # during your test run based on your test's specific device.
    # For more information about which environment variables are set and how they're
set, please see
    # https://docs.aws.amazon.com/devicefarm/latest/developerguide/custom-test-
environment-variables.html
```

```

- |-
appium --base-path=$APPIUM_BASE_PATH --log-timestamp \
  --log-no-colors --relaxed-security --default-capabilities \
  "{\"appium:deviceName\": \"\$DEVICEFARM_DEVICE_NAME\", \
  \"platformName\": \"\$DEVICEFARM_DEVICE_PLATFORM_NAME\", \
  \"appium:app\": \"\$DEVICEFARM_APP_PATH\", \
  \"appium:udid\": \"\$DEVICEFARM_DEVICE_UDID\", \
  \"appium:platformVersion\": \"\$DEVICEFARM_DEVICE_OS_VERSION\", \
  \"appium:chromedriverExecutableDir\":
  \"\$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR\", \
  \"appium:automationName\": \"UiAutomator2\"}" \
  >> $DEVICEFARM_LOG_DIR/appium.log 2>&1 &

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

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

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

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

```



```
# If you need to run any commands to generating logs and reports on how your test
performed,
# we recommend adding them to this section.
post_test:
  commands:

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

Migration vers l'hôte de test Amazon Linux 2

Pour migrer les tests existants de l'ancien hôte vers le nouvel hôte Amazon Linux 2, développez de nouveaux fichiers de spécifications de test basés sur vos fichiers préexistants. L'approche recommandée consiste à commencer par les nouveaux fichiers de spécifications de test par défaut pour vos types de tests. Migrez ensuite les commandes pertinentes de votre ancien fichier de spécifications de test vers le nouveau, en enregistrant l'ancien fichier en tant que sauvegarde. Cela vous permet de tirer parti des spécifications par défaut optimisées pour le nouvel hôte tout en réutilisant votre code existant. Cela vous permet de bénéficier de tous les avantages du nouvel hôte configuré de manière optimale pour vos tests, tout en conservant vos spécifications de test existantes à titre de référence lorsque vous adaptez les commandes au nouvel environnement.

Les étapes suivantes peuvent être utilisées pour créer un nouveau fichier de spécifications de test Amazon Linux 2 tout en réutilisant les commandes de votre ancien fichier de spécifications de test :

1. Connectez-vous à la console Device Farm à l'[adresse https://console.aws.amazon.com/devicefarm](https://console.aws.amazon.com/devicefarm).
2. Accédez au projet Device Farm contenant vos tests d'automatisation.
3. Choisissez Créer un nouveau cycle de test dans le projet.
4. Choisissez une application et un package de test déjà utilisés pour votre framework de test.
5. Choisissez Exécuter votre test dans un environnement personnalisé.
6. Choisissez le fichier de spécifications de test que vous utilisez actuellement pour les tests sur l'ancien hôte de test dans le menu déroulant des spécifications de test.

7. Copiez le contenu de ce fichier et collez-le localement dans un éditeur de texte pour référence ultérieure.
8. Dans le menu déroulant des spécifications de test, remplacez votre sélection de spécifications de test par le fichier de spécifications de test par défaut le plus récent.
9. Choisissez Modifier pour accéder à l'interface d'édition des spécifications de test. Vous remarquerez que, dans les premières lignes du fichier de spécifications de test, il a déjà opté pour le nouvel hôte de test :

```
android_test_host: amazon_linux_2
```

10. Consultez la syntaxe de sélection des hôtes de test [ici](#) et les principales différences entre les hôtes de test [ici](#).
11. Ajoutez et modifiez de manière sélective les commandes de votre fichier de spécifications de test enregistré localement à partir de l'étape 6 dans le nouveau fichier de spécifications de test par défaut. Choisissez ensuite Enregistrer sous pour enregistrer le nouveau fichier de spécifications. Vous pouvez désormais planifier des tests sur l'hôte de test Amazon Linux 2.

Différences entre les nouveaux hôtes de test et les anciens

Lorsque vous modifiez votre fichier de spécifications de test pour utiliser l'hôte de test Amazon Linux 2 et que vous transférez vos tests depuis l'ancien hôte de test, tenez compte de ces principales différences d'environnement :

- Sélection des versions logicielles : dans de nombreux cas, les versions logicielles par défaut ont changé. Par conséquent, si vous ne sélectionnez pas explicitement votre version logicielle dans l'hôte de test Legacy auparavant, vous souhaitez peut-être la spécifier maintenant sur l'hôte de test Amazon Linux 2 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 vous 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`, `vm`, et `rvm` 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 votre logiciel, vous sélectionnez les versions figurant dans la [liste des versions prises en charge](#).

- Tous les chemins de fichier codés en dur dans votre fichier de spécifications de test d'hôte Legacy sous forme de chemins absolus ne fonctionneront probablement pas comme prévu sur l'hôte de test Amazon Linux 2 ; 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. De plus, notez que la plupart des fichiers binaires dont vous avez besoin pour votre test se trouvent dans le PATH de l'hôte afin qu'ils soient immédiatement exécutables à partir du fichier de spécifications en utilisant uniquement leur nom (comme `appium`).
- La collecte de données de performance n'est pas prise en charge sur le nouvel hôte de test pour le moment.
- Version du système d'exploitation : l'ancien hôte de test était basé sur le système d'exploitation Ubuntu, tandis que le nouveau était basé sur Amazon Linux 2. Par conséquent, les utilisateurs peuvent remarquer certaines différences entre les bibliothèques système disponibles et les versions des bibliothèques système.
- 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. Pour plus d'informations, consultez [Working with Appium et AWS Device Farm](#).
- Pour les utilisateurs d'Appium, la variable d'`$DEVICEFARM_CHROMEDRIVER_EXECUTABLE` environnement a été supprimée au profit d'une nouvelle approche permettant aux clients d'accéder à Chromedriver pour Android. Consultez notre [fichier de spécifications de test par défaut](#) pour un exemple, qui utilise une nouvelle variable `$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR` d'environnement.

Note

Nous vous recommandons vivement de conserver telle quelle la commande du serveur Appium existante du fichier de spécifications de test par défaut.

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.

Variables d'environnement

Les variables d'environnement représentent les valeurs qui sont utilisées par vos tests automatisés. Vous pouvez utiliser ces variables d'environnement dans vos fichiers YAML et votre code de test. Dans un environnement de test personnalisé, Device Farm remplit dynamiquement les variables d'environnement lors de l'exécution.

Rubriques

- [Variables d'environnement communes](#)
- [Variables d'environnement Appium Java JUnit](#)
- [Variables d'environnement Appium Java TestNG](#)
- [Variables d'environnement Xcuitest](#)

Variables d'environnement communes

Tests sur Android

Cette section décrit les variables d'environnement personnalisées communes aux tests de plateforme Android pris en charge par Device Farm.

\$DEVICEFARM_DEVICE_NAME

Nom de l'appareil sur lequel vous exécutez les tests. Il représente l'identifiant unique (UDID) de l'appareil.

\$DEVICEFARM_DEVICE_PLATFORM_NAME

Nom de la plateforme de l'appareil. Il s'agit d'Android ou d'iOS.

\$DEVICEFARM_DEVICE_OS_VERSION

Version du système d'exploitation de l'appareil.

\$DEVICEFARM_APP_PATH

Chemin de l'application mobile sur la machine hôte où les tests sont en cours d'exécution. Le chemin de l'application est disponible uniquement pour les applications mobiles.

\$DEVICEFARM_DEVICE_UDID

Identifiant unique de l'appareil mobile qui exécute le test automatisé.

\$DEVICEFARM_LOG_DIR

Chemin des fichiers journaux générés au cours du test. Par défaut, tous 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_CHROMEDRIVER_EXECUTABLE_DIR

L'emplacement d'un répertoire contenant les exécutables Chromedriver nécessaires à une utilisation dans les tests Web et hybrides d'Appium.

\$ANDROID_HOME

Le chemin d'accès au répertoire d'installation du SDK Android.

Note

La variable d'ANDROID_HOME environnement est uniquement disponible sur l'hôte de test Amazon Linux 2 pour Android.

Tests iOS

Cette section décrit les variables d'environnement personnalisées communes aux tests de plateforme iOS pris en charge par Device Farm.

\$DEVICEFARM_DEVICE_NAME

Nom de l'appareil sur lequel vous exécutez les tests. Il représente l'identifiant unique (UDID) de l'appareil.

\$DEVICEFARM_DEVICE_PLATFORM_NAME

Nom de la plateforme de l'appareil. Il s'agit d'Android ou d'iOS.

\$DEVICEFARM_APP_PATH

Chemin de l'application mobile sur la machine hôte où les tests sont en cours d'exécution. Le chemin de l'application est disponible uniquement pour les applications mobiles.

\$DEVICEFARM_DEVICE_UDID

Identifiant unique de l'appareil mobile qui exécute le test automatisé.

\$DEVICEFARM_LOG_DIR

Chemin des fichiers journaux générés au cours du test.

\$DEVICEFARM_SCREENSHOT_PATH

Chemin des captures d'écran, le cas échéant, capturées au cours du test.

Variables d'environnement Appium Java JUnit

Cette section décrit les variables d'environnement utilisées par les tests Appium Java JUnit dans un environnement de test personnalisé.

\$DEVICEFARM_TESTNG_JAR

Chemin du fichier TestNG.jar.

\$DEVICEFARM_TEST_PACKAGE_PATH

Chemin du contenu décompressé du fichier du package de test.

Variables d'environnement Appium Java TestNG

Cette section décrit les variables d'environnement utilisées par les tests Appium Java TestNG dans un environnement de test personnalisé.

\$DEVICEFARM_TESTNG_JAR

Chemin du fichier TestNG.jar.

\$DEVICEFARM_TEST_PACKAGE_PATH

Chemin du contenu décompressé du fichier du package de test.

Variables d'environnement Xcuitest

\$DEVICEFARM_XCUITESTRUN_FILE

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.

Migration des tests d'un environnement de test standard vers un environnement de test personnalisé

Le guide suivant explique comment passer d'un mode d'exécution de test standard à un mode d'exécution personnalisé. 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.

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 de packaging 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.

4. **Obsolète** : les types de tests suivants seront déconseillés d'ici fin décembre 2023 dans le mode d'exécution standard :
 - Appium (toutes les langues)
 - Calabash
 - XCTest
 - UI Automation
 - UI Automator

- Tests sur le Web
- Explorateur intégré

Une fois obsolètes, vous ne pourrez plus utiliser ces frameworks en mode standard. Vous pouvez plutôt utiliser le mode personnalisé pour les types de tests listés ci-dessus.

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/](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 `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.

XCUITest iOS

Vous n'avez pas besoin d'effectuer de modifications pour déplacer vos tests XCUITest iOS vers un environnement de test personnalisé.

Extension des environnements de test personnalisés dans Device Farm

Le mode Device Farm Custom 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.

Configuration d'un code PIN

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.

Accélérer les tests basés sur Appium 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 `usePrebuiltWDA` fonctionnalité d'une suite de tests iOS afin d'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 basées sur XPath 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` moment du 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 `surhttp://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.

Utilisation de webhooks et d'autres API après l'exécution de vos tests

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](#).

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 API. 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 que votre suite de tests est 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 des API tierces 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.

Ajouter des fichiers supplémentaires à votre package de test

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 façon dont les packages de test iOS sont résignés. AWS Device Farm 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 packages XcuiTest, ajoutez des fichiers supplémentaires au répertoire se terminant par .app à l'intérieur du répertoire Payload. 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 seront conservés à l'emplacement où se trouve la variable d'environnement `$DEVICEFARM_TEST_PACKAGE_PATH`. (Cela signifie que si vous ajoutiez un fichier appelé `extra_file` à la racine de l'archive comme dans le premier exemple, il se trouverait dans `$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 fichier `testng.xml` 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 lui-même se trouve dans `$DEVICEFARM_TEST_PACKAGE_PATH`. 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 répertoire `/tmp` :

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

Dans le cas d'un fichier APK ou JAR, vos fichiers supplémentaires seront décompressés dans le répertoire `/tmp` (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 dans le répertoire *Payload*. Par exemple, sur la base de l'exemple IPA ci-dessus, le fichier se trouverait à l'emplacement */tmp/payload/AdfiOS UITests-Runner.app/extra_file* (référençable sous la forme */tmp/payload/ReferenceApp *.app/extra_file*).

Utilisation de l'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.

Rubriques

- [Création d'une session d'accès à distance dans AWS Device Farm](#)
- [Utiliser une session d'accès à distance dans AWS Device Farm](#)
- [Obtenez les 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](#).

- [Prérequis](#)
- [Création d'un test \(console\)](#)
- [Étapes suivantes](#)

Prérequis

- 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 avec la console Device Farm

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 Tests d'appareils mobiles, puis choisissez Projets.
3. Si vous avez déjà un projet, sélectionnez-le dans la liste. Sinon, créez un projet en suivant les instructions de [Création d'un projet dans AWS Device Farm](#).
4. Sous l'onglet Remote access (Accès à distance), choisissez Start a new session (Démarrer une nouvelle session).
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. Vous pouvez effectuer une recherche par :
 - Nom
 - Plateforme
 - Format
 - Type de flotte
6. Dans Session name (Nom de session), entrez le nom de la session.
7. Choisissez Confirm and start session (Confirmer et démarrer la session).

Étapes suivantes

Device Farm démarre la session dès que l'appareil demandé est disponible, généralement en quelques minutes. L'appareil demandé une boîte de dialogue apparaît jusqu'au démarrage de la session. Pour annuler la demande de session, choisissez `Cancel request` (Annuler la demande).

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.

Après le démarrage de la session, vous pouvez interagir avec l'appareil dans le navigateur Web.

Utiliser 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](#).

- [Prérequis](#)
- [Utiliser une session dans la console Device Farm](#)
- [Étapes suivantes](#)
- [Trucs et astuces](#)

Prérequis

- 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 `Temps restant` champ situé à côté du nom de l'appareil.

Installation d'une application

Pour installer une application sur le périphérique de session, dans `Installer des applications`, sélectionnez `Choisissez 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

AWS Device Farm n'affiche pas de confirmation après l'installation d'une application. Essayez d'interagir avec l'icône de l'application pour voir si l'application est prête à l'emploi. Lorsque vous chargez une application, l'application devient parfois disponible après un certain délai. Consultez la barre d'état système pour déterminer si l'application est disponible.

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 Applications récentes.

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 plus d'informations, veuillez consulter [Obtenir les résultats des sessions](#).

Trucs et astuces

Il est possible que vous soyez confronté à des problèmes de performances avec la session d'accès à distance dans certaines régions AWS. 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.

Obtenez les résultats d'une session d'accès à distance dans AWS Device Farm

Pour plus d'informations sur les sessions, consultez [Séances](#).

- [Prérequis](#)
- [Afficher les détails de la session](#)
- [Téléchargement de vidéos ou de journaux de session](#)

Prérequis

- Terminez une session. Suivez les instructions de [Utiliser 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 Tests d'appareils mobiles, puis choisissez Projets.
2. Choisissez le projet contenant la session.
3. Choisissez Accès à distance, puis choisissez 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.

Utilisation d'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. 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 [Gestion des appareils privés 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](#).

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

- [Gestion des appareils privés dans AWS Device Farm](#)
- [Sélection d'appareils privés dans un pool d'appareils](#)
- [Ignorer la nouvelle signature d'une application sur des appareils privés dans AWS Device Farm](#)
- [Utilisation des services de point de terminaison Amazon VPC avec Device Farm](#)
- [Travailler avec Amazon VPC sur l'ensemble du siteAWS Régions](#)
- [Résiliation d'appareils privés](#)

Gestion des 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 propre à votre compte AWS.

Note

À l'heure actuelle, les appareils privés sont disponibles dans la région AWS ouest des États-Unis (Oregon) (us-west-2) uniquement.

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.

Cette rubrique explique comment créer un profil d'instance et effectuer d'autres tâches courantes de gestion des appareils.

Rubriques

- [Création d'un profil d'instance](#)
- [Gestion d'une instance d'appareil privée](#)
- [Création d'un test ou démarrage d'une session d'accès à distance](#)
- [Étapes suivantes](#)

Création d'un profil d'instance

Pour contrôler le comportement des appareils privés lors d'un test ou d'une session d'accès à distance, vous pouvez créer ou modifier un profil d'instance dans Device Farm. Vous n'avez pas besoin d'un profil d'instance pour commencer à utiliser vos appareils privés.

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 **Test d'appareils mobiles**, puis choisissez **Appareils privés**.
3. Choisissez **Instance profiles (Profils d'instance)**.

4. ChoisissezCréation d'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 après chaque test ou chaque fin de session :
 - Redémarrer après utilisation— Pour redémarrer l'appareil, cochez cette case. Par défaut, cette case est décochée (`false`).
 - Nettoyage du colis— 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, sélectionnez le Nettoyage du colis à cocher, puis choisissez 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 Save (Enregistrer).

Gestion d'une instance d'appareil privée

Si vous avez déjà un ou plusieurs appareils privés dans votre parc, vous pouvez afficher des informations sur chaque instance d'appareil et en gérer certains paramètres. Vous pouvez également demander une instance d'appareil privé supplémentaire.

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 Test d'appareils mobiles, puis choisissez Appareils privés.
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. (Facultatif) Pour demander une nouvelle instance de périphérique privée, choisissez Demander une instance de périphérique 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 à côté de l'instance dont vous souhaitez consulter les informations ou que vous souhaitez 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

- (Facultatif) Pour Profile (Profil), choisissez un profil d'instance à attacher à l'instance de l'appareil. Ceci peut être utile si vous souhaitez toujours exclure un package d'applications spécifique des tâches de nettoyage, par exemple.
- (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.
- Choisissez Save (Enregistrer).

Création d'un test ou démarrage d'une session d'accès à distance

Une fois que vous avez configuré un parc d'appareils privés, vous pouvez créer des séries de tests ou démarrer des sessions d'accès à distance avec un ou plusieurs appareils privés dans votre parc.

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 **Test d'appareils mobiles**, puis choisissez **Projets**.
3. Choisissez un projet existant dans la liste ou créez-en un nouveau. Pour créer un nouveau projet, choisissez **Nouveau projet**, entrez le nom du 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 le **Sélectionnez les appareils** étape, vous pouvez modifier un pool d'appareils existant ou créer un nouveau pool d'appareils qui inclut uniquement les appareils privés que l'équipe Device Farm a configurés et associés à votre **AWS** compte. Pour plus d'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 le **Choisissez un appareil** page, sélectionnez **Instances d'appareils privés uniquement** 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 |
|-----------------------|-----------------------|-----------|----------|----|-------------|-------------|--------|
| <input type="radio"/> | OnePlus 8T | AVAILABLE | Android | 11 | Phone | - | - |
| <input type="radio"/> | Samsung Galaxy Tab S7 | AVAILABLE | Android | 11 | Tablet | - | - |

Étapes suivantes

Une fois que vous avez configuré vos appareils privés, vous pouvez également gérer vos appareils privés en procédant de l'une des manières suivantes :

- [Ignorer la resignature d'application sur les appareils privés](#)
- [Utilisez les services de point de terminaison Amazon Virtual Private Cloud avec Device Farm](#)

Pour supprimer un profil d'instance, sur le Profils d'instancemenu, choisissez l'option de bascule à côté de l'instance que vous souhaitez supprimer, puis choisissez Supprimer.

Sélection d'appareils privés dans un pool d'appareils

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, AWS Interface de ligne de commande (AWSCLI) ou l'API

Device Farm pour créer ou modifier un pool de périphériques 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 aviez trois *iPhone 13 d'Apple* appareils sur la version *iOS16,10* 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 efficace 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 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 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, voir [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 d'appareils privés sont représentées par cette étiquette d'instance d'appareil et, parmi ces instances, sélectionne au hasard celle qui est 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

Un ARN d'instance de périphérique est un identifiant représentant une instance physique de périphérique bare metal déployée dans un parc privé. Par exemple, si vous aviez trois *iPhone 13* appareils sous système d'exploitation *15,0.0* dans votre parc privé, alors que chaque appareil partagerait le même ARN, chaque appareil aurait 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 ARN et les étiquettes des instances d'appareils ne correspondent pas à votre cas d'utilisation. Les ARN des instances de périphériques sont souvent utilisés 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 celle-ci. 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. Ainsi, si vous utilisez les ARN des instances d'appareils 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 son test, cela peut constituer un flux de travail efficace (par rapport aux ARN du périphérique). 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 de périphérique 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 de périphérique. Si cette instance d'appareil n'est pas disponible, Device Farm mettra le test en file d'attente sur l'appareil une fois 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 d'appareils contenant des règles pour les ARN des appareils et les ARN des instances de périphériques), vous devez créer le pool via l'interface de ligne de commande 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 Tests d'appareils mobiles, puis choisissez Projets.
3. Choisissez un projet existant dans la liste ou créez-en un nouveau. Pour créer un nouveau projet, choisissez Nouveau projet, entrez le nom du projet, puis choisissez Soumettre.
4. Choisissez Automated tests (Tests automatisés), puis choisissez Create a new run (Créer une exécution). L'assistant vous décrit la marche à suivre pour choisir votre application et configurer le test que vous souhaitez exécuter.
5. Pour le Sélectionnez les appareils étape, choisissez Création d'un pool d'appareils, et 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 Création d'un pool de périphériques statique, puis sélectionnez les types de périphériques spécifiques dans la liste que vous souhaitez utiliser dans le pool de périphériques. Ne pas sélectionner Instances d'appareils privés uniquement car cette option entraîne la création du pool de périphériques 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 |
|------|-----------|----------|----|-------------|-------------|--------|
| ⌂ | Available | Android | 10 | Phone | | - |

Cancel Create

- b. Pour utiliser les règles d'étiquetage des instances d'appareils pour votre pool d'appareils, choisissezCréation d'un pool d'appareils dynamique. Ensuite, pour chaque étiquette que vous souhaitez utiliser dans le pool d'appareils, choisissezAjouter une règle. Pour chaque règle, sélectionnezÉtiquettes d'instanceen tant queField, choisissezContienten tant queOperator, et spécifiez l'étiquette d'instance de périphérique de votre choix en tant queValue.

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

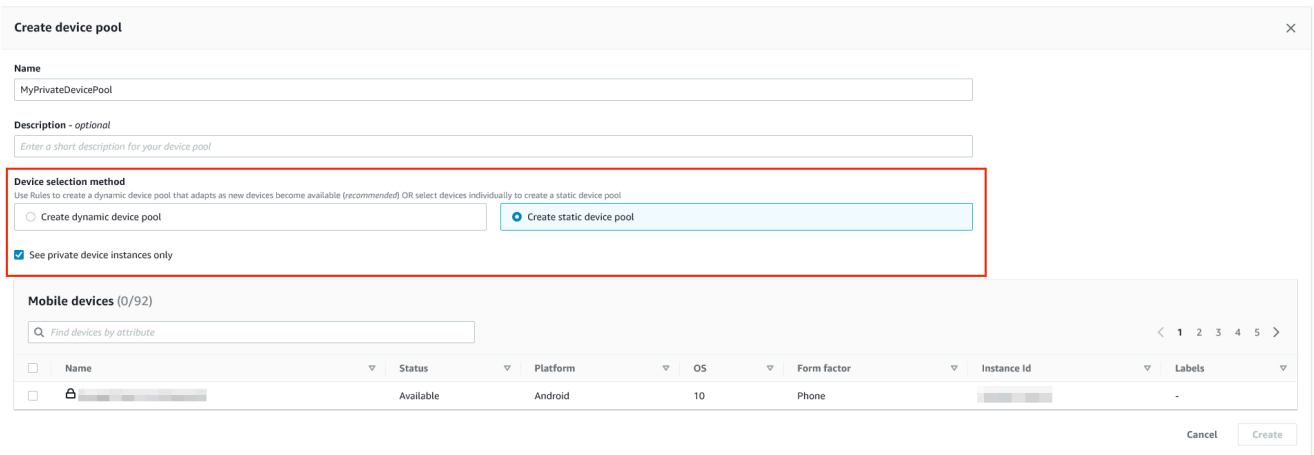
Mobile devices (0/92)

Find devices by attribute

| Name | Status | Platform | OS | Form factor | Instance Id | Labels |
|------|--------|----------|----|-------------|-------------|--------|
|------|--------|----------|----|-------------|-------------|--------|

Cancel Create

- c. Pour utiliser les règles ARN des instances de périphériques pour votre pool d'appareils, choisissezCréation d'un pool de périphériques statique, puis sélectionnezInstances d'appareils privés uniquementpour limiter la liste des appareils aux seules instances d'appareils privés que Device Farm a associées à votreAWScompte.



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

| <input type="checkbox"/> | Name | Status | Platform | OS | Form factor | Instance Id | Labels |
|--------------------------|------|-----------|----------|----|-------------|-------------|--------|
| <input type="checkbox"/> | Ⓐ | Available | Android | 10 | Phone | | - |

Cancel Create

6. Sélectionnez Create (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 AWS CLI, voir [Référence AWS CLI](#).

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, voir [Automatiser un parc d'appareils](#).

Ignorer la nouvelle signature d'une application sur des appareils privés dans AWS Device Farm

Lorsque vous utilisez des appareils privés, vous pouvez ignorer l'étape au cours de laquelle AWS Device Farm resigne 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 la fonctionnalité de votre application est interrompue 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 [FAQ sur AWS Device Farm](#).

Pour ignorer la resignature des applications pour une série de tests, sélectionnez Skip app re-signing (Ignorer la resignature d'application) sur la page Configurer lorsque vous créez la série de tests.

Configure

Setup test framework

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

Built-in: Fuzz

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

Event count

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

6000

Event throttle

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

50

Randomizer seed

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

Enter a randomizer seed

▼ Advanced Configuration (optional)

Configuration specific to Private Devices

App re-signing

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

Skip app re-signing

Other Configuration

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

Video recording

If checked, enables video recording during test execution.

Enable video recording

Note

Si vous utilisez l'infrastructure XCTest, l'option Skip app re-signing (Ignorer la resignature d'application) n'est pas disponible. Pour plus d'informations, veuillez consulter [Utilisation de XCTest pour iOS et AWS Device Farm](#).

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, ID et profils de la console.
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` de 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 pouvez installer l'application que vous souhaitez tester sur l'appareil privé, ou vous pouvez installer une application factice signée avec le même certificat que l'application que vous souhaitez tester. L'installation d'une application factice signée avec le même certificat présente un avantage. En effet, une fois que vous avez approuvé le profil de configuration ou le développeur d'applications d'entreprise, toutes les applications de ce développeur sont approuvées sur l'appareil privé jusqu'à ce que vous les supprimiez. Par conséquent, lorsque vous chargez une nouvelle version de l'application que vous souhaitez tester, vous n'avez pas à approuver à nouveau le développeur d'applications. 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.

Avant de démarrer votre session d'accès à distance, suivez les étapes décrites dans [Création d'un profil d'instance](#) pour créer ou modifier un profil d'instance dans Device Farm. Dans le profil de l'instance, ajoutez l'ID de bundle de l'application de test ou de l'application fictive au **Exclure les packages du nettoyage** réglage. 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 charger l'application factice sur l'appareil à l'aide d'une session d'accès à distance, ce qui vous permet de lancer l'application et d'approuver le 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.

N'oubliez pas d'ajouter également l'application fictive ou l'application que vous souhaitez tester dans Exclure les packages du nettoyage paramètre pour le profil d'instance attaché à cette instance.

2. Lorsque votre session à distance démarre, choisissez un fichier pour installer une application qui utilise votre profil de provisionnement interne.
3. Lancez l'application que vous venez de charger.
4. Suivez les instructions pour approuver le certificat de développeur.

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.

Utilisation des services de point de terminaison Amazon VPC avec Device Farm

📘 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 [contacter nous](#).

Amazon Virtual Private Cloud (Amazon VPC) est un AWS service que vous pouvez utiliser pour lancer AWS les ressources d'un réseau virtuel que vous définissez. Avec un VPC, vous pouvez contrôler les paramètres de votre 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 l'ouest des États-Unis (Oregon) (us-west-2) AWS Région, 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 activer votre compte AWS pour utiliser cette fonctionnalité avec les 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, voir [Services de point de terminaison VPC \(AWS\)PrivateLink](#) dans le AWS PrivateLink Guide.

Important

La fonctionnalité de point de terminaison Device Farm VPC vous permet de connecter en toute sécurité les services internes privés de votre VPC au VPC public Device Farm en utilisant AWS PrivateLink connexions. Bien que la connexion soit sécurisée et privée, la sécurité dépend de votre protection de vos informations d'identification AWS. Si vos informations d'identification AWS sont compromises, un pirate informatique peut accéder aux données de vos services ou les divulguer 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 dans l'ouest des États-Unis (Oregon) (us-west-2) Région, 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. L'équilibreur de charge réseau associé à un service de point de terminaison VPC détermine les zones de disponibilité pour ce service de point de terminaison VPC. Si le service de point de terminaison de votre 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 le nom de ressource Amazon (ARN) du point de terminaison Device Farm VPC (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](#).

De plus, si vous conservez le paramètre `Acceptation 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 des terminaux`. 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 équilibreur de charge réseau


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 équilibreur de charge réseau 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 `Équilibreurs de charge`.
3. Choisissez `Créer un équilibreur de charge`.

4. Sous **Équilibreur de charge réseau**, choisissez **Créer**.
5. Sur la **Création d'un équilibreur de charge réseau** page, sous **Configuration de base**, procédez comme suit :
 - a. Entrez un **équilibreur de charge** Nom.
 - b. Pour **Schéma**, choisissez **Interne**.
6. Sous **Network mapping (Mappage réseau)**, procédez comme suit :
 - a. Choisissez le **VPC** pour votre groupe cible.
 - b. Sélectionnez les éléments suivants **Cartographies**:
 - us-west-2a
 - us-west-2b
 - us-west-2c
7. Sous **Écouteurs et routage**, utilisez le **Protocole** et **Port options** pour choisir votre groupe cible.

 **Note**

Par défaut, l'équilibrage de charge des zones de disponibilité croisée est désactivé. Parce que l'équilibreur de charge utilise les zones de disponibilité us-west-2a, us-west-2b, et us-west-2c, soit les cibles doivent être enregistrées dans chacune de ces zones de disponibilité, soit, si vous enregistrez des cibles dans moins de trois zones, vous devez activer l'équilibrage de charge entre zones. 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 équilibreur de charge réseau 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 **équilibres de charge**.
3. Choisissez **Créer un équilibreur de charge**.
4. Sous **Équilibreur de charge réseau**, choisissez **Créer**.

5. Sur leConfigurer l'équilibreur de chargepage, sousConfiguration de base, procédez comme suit :
 - a. Entrez un équilibreur de chargeNom.
 - b. PourSchéma, choisissezInterne.
6. SousAuditeurs, sélectionnez leProtocoleetPortque votre groupe cible utilise.
7. SousZones de disponibilité, procédez comme suit :
 - a. Choisissez leVPCpour votre groupe cible.
 - b. Sélectionnez les éléments suivantsZones de disponibilité:
 - us-west-2a
 - us-west-2b
 - us-west-2c
 - c. ChoisissezSuivant : configurer les paramètres de sécurité.
8. (Facultatif) Configurez vos paramètres de sécurité, puis choisissezSuivant : configurer le routage.
9. Sur la page Configure Routing, procédez de la façon suivante :
 - a. Pour Groupe cible, choisissez Groupe cible existant.
 - b. PourNom, choisissez votre groupe cible.
 - c. ChoisissezSuivant : enregistrer les cibles.
10. Sur leEnregistrer les ciblespage, passez en revue vos cibles, puis choisissezSuivant : revue.

 Note

Par défaut, l'équilibrage de charge des zones de disponibilité croisée est désactivé. Parce que l'équilibreur de charge utilise les zones de disponibilitéus-west-2a,us-west-2b, etus-west-2c, soit les cibles doivent être enregistrées dans chacune de ces zones de disponibilité, soit, si vous enregistrez des cibles dans moins de trois zones, vous devez activer l'équilibrage de charge entre zones. 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 choisissezCréez.

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 Services pour terminaux.
3. Choisissez Create endpoint service (Créer un service de point de terminaison).
4. Effectuez l'une des actions suivantes :
 - Si vous possédez déjà un équilibreur de charge réseau que vous souhaitez que le service de point de terminaison utilise, choisissez-le sous Équilibreurs de charge disponibles, puis passez à l'étape 5.
 - Si vous n'avez pas encore créé d'équilibreur de charge réseau, choisissez Créer un nouvel équilibreur de charge. La console Amazon EC2 s'ouvre. Suivez les étapes décrites dans [Création d'un équilibreur de charge réseau](#) en commençant par l'étape 3, puis poursuivez avec ces étapes dans la console Amazon VPC.
5. Pour Zones de disponibilité incluses, vérifiez que us-west-2a, us-west-2b, 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 Réglages supplémentaires, transparent 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. Sélectionnez Create (Créer).
8. Dans le nouveau service de point de terminaison, choisissez Autoriser les directeurs.
9. [Contactez-nous](#) pour obtenir l'ARN du point de terminaison Device Farm VPC (ARN du service) à ajouter à la liste des autorisations pour le service du point de terminaison, puis ajouter 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 **Test d'appareils mobiles**, puis **Appareils privés**.
3. Choisissez **Configurations VPCE**.
4. Choisissez **Création d'une configuration VPCE**.
5. Sous **Création d'une nouvelle configuration VPCE**, entrez un **Nom** pour la configuration du point de terminaison du VPC.
6. Pour **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. Pour **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 **Save (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 du 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](#).

Travailler avec Amazon VPC sur l'ensemble du siteAWSRégions

Les services Device Farm sont situés uniquement dans l'ouest des États-Unis (Oregon) (us-west-2) Région. Vous pouvez utiliser Amazon Virtual Private Cloud (Amazon VPC) pour accéder à un service de votre Amazon Virtual Private Cloud depuis un autreAWSRégion utilisant Device Farm. Si Device Farm et votre service se trouvent dans la même région, voir [Utilisation des services de point de terminaison Amazon VPC avec Device Farm](#).

Il existe deux manières d'accéder à vos services privés situés dans une région différente. Si vous avez des services situés dans une autre région, ce n'est pas le casus-west-2, vous pouvez utiliser le peering VPC pour associer le VPC de cette région à un autre VPC qui s'interface avec Device Farm dansus-west-2. Toutefois, si vous avez des services dans plusieurs régions, une passerelle de transit vous permettra d'accéder à ces services avec une configuration réseau plus simple.

Pour plus d'informations, voir [Scénarios de peering VPC](#) dans le Guide de peering d'Amazon VPC.

Appairage de VPC

Vous pouvez associer deux VPC situés dans des régions différentes, à condition qu'ils 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 des VPC de s'adresser les unes 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 (appeléVPC-1) est situé dans l'ouest des États-Unis (Oregon) (us-west-2) Région. Le deuxième VPC dans cet exemple (appeléVPC-2) se trouve dans une autre région.

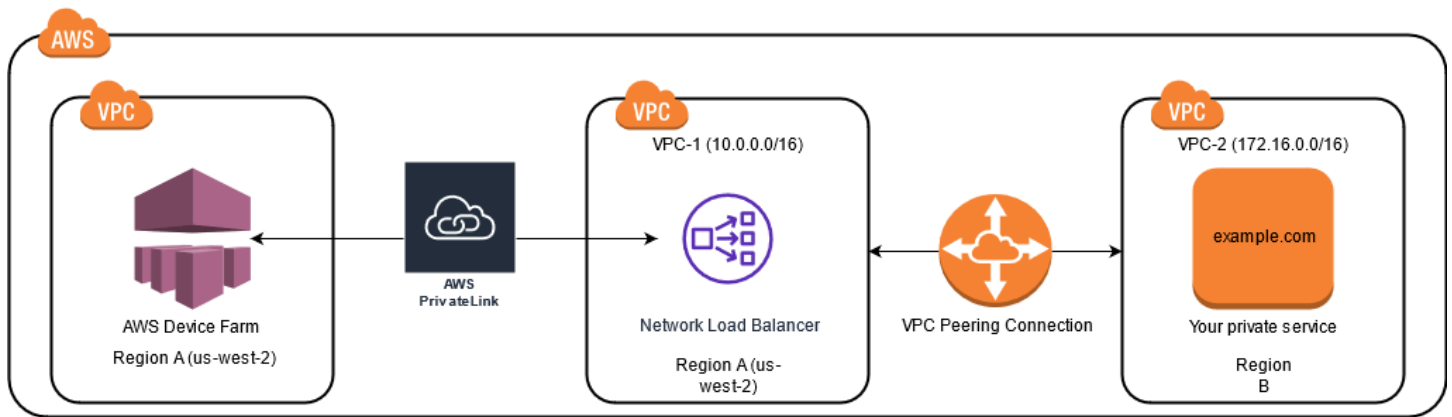
Exemple interrégional de Device Farm VPC

| Composant du VPC | VPC-1 | VPC-2 |
|------------------|-------------|---------------|
| CIDR | 10.0.0.0/16 | 172.16.0.0/16 |

Important

L'établissement d'une connexion d'appairage entre deux VPC peut modifier la posture de sécurité des VPC. En outre, l'ajout de nouvelles entrées à leurs tables de routage peut modifier le niveau de sécurité des ressources au sein des VPC. 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

- [Prérequis](#)
- [Étape 1 : configurer une connexion de peering entre VPC-1 et VPC-2](#)
- [Étape 2 : Mettre à jour les tables de routage dans VPC-1 et VPC-2](#)
- [Étape 3 : Création d'un groupe cible](#)
- [Étape 4 : Création d'un équilibreur de charge réseau](#)
- [Étape 5 : créer un service de point de terminaison VPC](#)
- [Étape 6 : Création d'une configuration de point de terminaison VPC dans Device Farm](#)
- [Étape 7 : Création d'un essai](#)
- [Créez un réseau évolutif avec Transit Gateway](#)

Prérequis

Cet exemple nécessite les éléments suivants :

- Deux VPC configurés avec des sous-réseaux contenant des blocs CIDR ne se chevauchant pas.
- VPC-1 doit être dans leus-west-2 Sous-réseaux régionaux et contenant pour les zones de disponibilité us-west-2a, us-west-2b, et us-west-2c.

Pour plus d'informations sur la création de VPC et la configuration de sous-réseaux, voir [Utilisation de VPC et de sous-réseaux](#) dans le Guide de peering d'Amazon VPC.

Étape 1 : configurer une connexion de peering entre VPC-1 et VPC-2

Établissez une connexion d'appairage entre les deux VPC contenant des blocs CIDR ne se chevauchant pas. Pour ce faire, voir [Création et acceptation de connexions d'appairage VPC](#) dans le Guide de peering d'Amazon VPC. En utilisant le scénario interrégional de cette rubrique et le Guide de peering d'Amazon VPC, l'exemple de configuration de connexion d'appairage suivant est créé :

Name (Nom)

Device-Farm-Peering-Connection-1

ID VPC (demandeur)

vpc-0987654321gfedcba (VPC-2)

Compte

My account

Région

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

Identifiant VPC (Accepteur)

vpc-1234567890abcdefg (VPC-1)

Note

Assurez-vous de consulter les quotas de connexion d'appairage de votre VPC lorsque vous établissez de nouvelles connexions d'appairage. Pour plus d'informations, veuillez consulter [Quotas Amazon VPC](#) dans le Guide de peering d'Amazon VPC.

Étape 2 : Mettre à jour les tables de routage dans VPC-1 et VPC-2

Après avoir configuré une connexion d'appairage, vous devez établir une route de destination entre les deux VPC pour transférer des données entre eux. Pour établir cet itinéraire, vous pouvez mettre à jour manuellement la table de routage de VPC-1 pour pointer vers le sous-réseau de VPC-2 et vice versa. Pour ce faire, voir [Mettez à jour vos tables de routage pour une connexion d'appairage VPC](#) dans le Guide de peering d'Amazon VPC. En utilisant le scénario interrégional de cette rubrique

et le Guide de peering d'Amazon VPC, 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 équilibreur de charge réseau 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 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 (comme un équilibreur de charge réseau) 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 <https://console.aws.amazon.com/ec2/>, puis choisissez Instances.
- 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, voir [Associer un équilibreur de charge à votre groupe Auto Scaling](#) dans le Guide de l'utilisateur d'Amazon EC2 Auto Scaling.

Vous pouvez ensuite ouvrir la console Amazon EC2 à l'adresse <https://console.aws.amazon.com/ec2/>, puis choisissez Interfaces réseau. À partir de là, vous pouvez afficher 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, voir [Créez un groupe cible pour votre Network Load Balancer](#) dans le Guide de l'utilisateur pour les équilibreurs de charge réseau.

Les groupes cibles pour les services d'un autre VPC nécessitent la configuration suivante :

- Pour Choisir un type de cible, choisissez Adresses IP.
- Pour VPC, choisissez le VPC qui hébergera l'équilibreur de charge. Pour l'exemple de sujet, ce sera VPC-1.
- Sur le Enregistrer les cibles page, enregistrez une cible pour chaque adresse IP dans VPC-2.

Pour Réseau, choisissez Autre adresse IP privée.

Pour Zone de disponibilité, choisissez les zones souhaitées dans VPC-1.

Pour Adresse IPv4, choisissez VPC-2 Adresse IP

Pour 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.

En utilisant le scénario interrégional de cette rubrique et le 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

Adresse IPv4

172.16.100.60

Ports

80

Étape 4 : Création d'un équilibreur de charge réseau

Créez un équilibreur de charge réseau en utilisant le groupe cible décrit dans [étape 3](#). Pour ce faire, voir [Création d'un équilibreur de charge réseau](#).

À l'aide du scénario interrégional de cette rubrique, les valeurs suivantes sont utilisées dans un exemple de configuration d'équilibreur de charge réseau :

Nom de l'équilibreur de charge

my-nlb

Schéma

Internal

VPC

vpc-1234567890abcdefg (VPC-1)

Mappage

us-west-2a - subnet-4i23iuufkdiufsloi

us-west-2b - subnet-7x989pkjj78nmn23j

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

Protocole/Port

```
TCP : 80
```

Groupe cible

```
my-target-group
```

Étape 5 : créer un service de point de terminaison VPC

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 infrastructure supplémentaire, telle qu'une passerelle Internet, une instance NAT ou une connexion VPN.

Pour ce faire, voir [Création d'un service de point de terminaison Amazon VPC](#).

Étape 6 : Création d'une configuration de point de terminaison VPC dans 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, voir [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 :

Name (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 essai

Vous pouvez créer des cycles de test qui utilisent la configuration du point de terminaison VPC décrite dans [étape 6](#). Pour plus d'informations, consultez [Création d'un test dans Device Farm](#) ou [Création d'une session](#).

Créez un réseau évolutif avec Transit Gateway

Pour créer un réseau évolutif utilisant plus de deux VPC, vous pouvez utiliser Transit Gateway pour faire office de hub de transit réseau afin d'interconnecter vos VPC et les réseaux sur site. Pour configurer un VPC dans la même région que Device Farm afin d'utiliser une passerelle de transit, vous pouvez suivre le [Services de point de terminaison Amazon VPC avec Device Farm](#) guide pour cibler les ressources dans 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 des passerelles de transit Amazon VPC.

Résiliation d'appareils privés

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.

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.

VPC-ENI dans AWS Device Farm

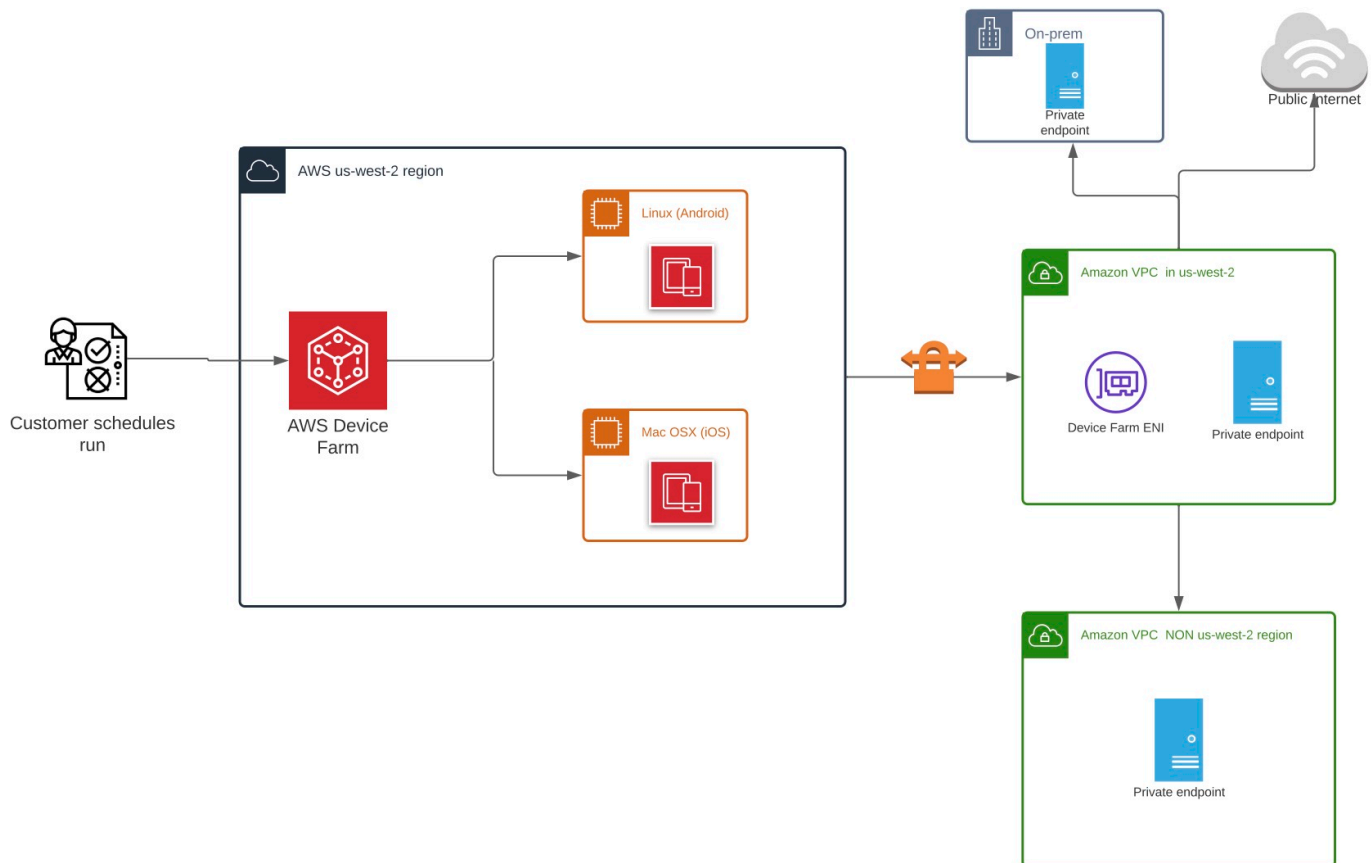
Warning

Cette fonctionnalité n'est disponible que sur [appareils privés](#). Pour demander l'utilisation d'un appareil privé sur votre AWS compte, s'il vous plaît [contactez nous](#). 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 AWS, un logiciel sur site ou un autre fournisseur de cloud.

Vous pouvez connecter les appareils mobiles Device Farm et leurs machines hôtes à un environnement Amazon Virtual Private Cloud (Amazon VPC) dans us-west-2 Région, qui permet d'accéder à des zones isolées, non-internet-facing services et applications via un [interface réseau élastique](#). Pour plus d'informations sur les VPC, consultez le [Guide de l'utilisateur d'Amazon VPC](#).

Si votre point de terminaison privé ou votre VPC ne se trouve pas dans us-west-2 Région, vous pouvez la lier à un VPC dans us-west-2 Région utilisant des solutions telles qu'un [Passerelle de transit](#) ou [Peering en PVC](#). Dans de telles situations, Device Farm créera une ENI dans un sous-réseau que vous fournissez à votre us-west-2 Région VPC, et vous serez chargé de vous assurer qu'une connexion peut être établie entre us-west-2 Le VPC de région et le VPC de l'autre région.



Pour plus d'informations sur l'utilisation AWS CloudFormation pour créer et associer automatiquement des VPC, consultez le [Modèles VPC Peering](#) dans le AWS CloudFormation référentiel de modèles sur GitHub.

Note

Device Farm ne facture rien pour la création d'ENI 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, des CDN publics) 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

- [AWSContrôle d'accès et IAM](#)
- [Rôles liés à un service](#)
- [Prérequis](#)
- [Connexion à Amazon VPC](#)
- [Limites](#)

AWSContrôle d'accès et IAM

AWS Device Farm vous permet d'utiliser [AWS Identity and Access Management](#) (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 :

```
{
  "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

Utilisations d'AWS Device Farm AWS Identity and Access Management (IAM) [rôles liés aux services](#). Un rôle lié à un service est un type unique de rôle IAM directement lié à Device Farm. Les rôles liés au 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 d'approbation 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 parc de périphériques, 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`— Permet à Device Farm d'accéder aux ressources AWS en votre nom.

Le rôle lié à un service `AWSServiceRoleForDeviceFarm` approuve les services suivants pour endosser 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
 - Décrire les VPC
 - 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 :

```
{
  "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 plus d'informations, consultez [Service-Linked Role Permissions \(autorisations du rôle lié à un service\)](#) dans le IAM User Guide (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, vous n'avez pas besoin de créer manuellement un rôle lié à un service. Lorsque vous créez votre première ressource Device Farm dans leAWS Management Console, leAWS CLI, ou leAWSAPI, 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 avecFerme d'appareilscas d'utilisation. Dans l'interface AWS CLI ou l'API AWS, créez un rôle lié à un service avec le nom de service `devicefarm.amazonaws.com`. 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 leAWSServiceRoleForDeviceFarmrôle lié au service. Une fois que vous avez 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 plus d'informations, consultez [Editing a Service-Linked Role](#) (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 fonction 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 à un service à l'aide d'IAM

Utilisez la console IAM, l'AWS CLI ou l'API AWS pour supprimer le rôle lié à un service `AWSServiceRoleForDeviceFarm`. Pour plus d'informations, veuillez consulter [Deleting a Service-Linked Role](#) (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 de plus amples informations, consultez [Regions and Endpoints 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 rôle `AWSServiceRoleForDeviceFarm` dans les régions suivantes :

| Nom de la région | Identité de la région | Assistance dans Device Farm |
|--------------------------------|-----------------------|-----------------------------|
| US East (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 |
| Asie-Pacifique (Osaka) | ap-northeast-3 | Non |
| Asie-Pacifique (Séoul) | ap-northeast-2 | Non |
| Asie-Pacifique (Singapour) | ap-southeast-1 | Non |

| Nom de la région | Identité de la région | Assistance dans Device Farm |
|-----------------------------|-----------------------|-----------------------------|
| 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 |

Prérequis

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 avoir un AWS compte, utilisateur ou rôle autorisé à créer un rôle lié au service. Lorsque vous utilisez des points de terminaison Amazon VPC avec les fonctionnalités de test mobile de Device Farm, Device Farm crée un AWS Identity and Access Management rôle lié au service (IAM).
- Device Farm peut se connecter aux VPC uniquement dans le us-west-2 Région. Si vous n'avez pas de VPC dans le us-west-2 Région, vous devez en créer une. Ensuite, pour accéder aux ressources d'un VPC d'une autre région, vous devez établir une connexion d'appairage entre le VPC du us-west-2 Région et VPC de l'autre région. Pour plus d'informations sur les VPC de peering, consultez le [Guide de peering d'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 dans leus-west-2 Région :
 - ID de VPC
 - ID de sous-réseau
 - ID de groupe de sécurité
- 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 sessions futures associées au projet utiliseront la connexion VPC configurée.
- Vous ne pouvez pas utiliser les configurations VPC-ENI avec l'ancienne fonctionnalité VPCE.
- Nous recommandons vivement 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 au VPC pour un projet, vous devez connaître les informations suivantes :

- L'ID du VPC dans leus-west-2 si votre application y est hébergée ou si leus-west-2 ID de VPC qui se connecte à un autre VPC d'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.

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 l'interface de ligne de commande 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 Tests d'appareils mobiles, puis choisissez Projets.
3. Sous Projets de tests mobiles, choisissez le nom de votre projet dans la liste.
4. Choisissez Project settings (Paramètres du projet).
5. Dans le Paramètres du cloud privé virtuel (VPC) section, vous pouvez modifier le VPC, Subnets, et Security Groups.
6. Choisissez Save (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
```

Limites

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 IPv4 disponibles.
- 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 un [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.
- 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.

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 entreprises 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 de CloudTrail événements relatifs à un compartiment Amazon S3, y compris des événements pour AWS Device Farm. Si vous ne configurez pas de journal d'activité, vous pouvez toujours afficher les événements les plus récents dans la console CloudTrail dans Historique des événements. En utilisant les informations collectées par CloudTrail, vous pouvez déterminer la demande envoyée à AWS Device Farm, l'adresse IP à partir de laquelle la demande a été effectuée, l'auteur de la demande, la date à laquelle elle a été faite, ainsi que des informations supplémentaires.

Pour en savoir plus sur CloudTrail, veuillez consulter le [Guide de l'utilisateur AWS CloudTrail](#).

Informations sur AWS Device Farm dans CloudTrail

CloudTrail est activé sur votre compte AWS lorsque vous créez le compte. Lorsqu'une activité a lieu dans AWS Device Farm, elle est enregistrée dans un CloudTrail événement avec d'autres AWS événements de service à Historique de l'événement. Vous pouvez afficher, rechercher et télécharger les événements récents dans votre compte AWS. Pour plus d'informations, consultez [Affichage des événements avec l'historique des événements CloudTrail](#).

Pour un enregistrement continu des événements survenus dans votre AWS compte, y compris les événements pour AWS Device Farm, créez un parcours. Un sentier permet CloudTrail pour envoyer des fichiers journaux dans 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 d'activité consigne les événements de toutes les régions dans la partition AWS et livre les fichiers journaux dans le compartiment Amazon S3 de votre choix. En outre, vous pouvez configurer d'autres services AWS pour analyser plus en profondeur les données d'événement collectées dans les journaux CloudTrail et agir sur celles-ci. Pour en savoir plus, consultez les ressources suivantes :

- [Présentation de la création d'un journal d'activité](#)
- [Intégrations et services supportés par CloudTrail](#)

- [Configuration des Notifications de Amazon SNS pour CloudTrail](#)
- [Réception de fichiers journaux CloudTrail de plusieurs régions](#) et [Réception de fichiers journaux CloudTrail de plusieurs comptes](#)

Quand CloudTrail la journalisation est activée dans votre AWS compte, les appels d'API effectués aux actions de Device Farm sont suivis dans des fichiers journaux. Les enregistrements de Device Farm sont écrits avec d'autres AWS les enregistrements de service dans un fichier journal. CloudTrail détermine quand créer un fichier et écrire dedans en fonction d'une période et d'une taille de fichier.

Toutes les actions de Device Farm sont enregistrées et documentées dans le [Référence AWS CLI](#) et le [Automatiser un parc d'appareils](#). Par exemple, les appels visant à créer un nouveau projet ou à exécuter dans Device Farm génèrent des entrées dans CloudTrail fichiers 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 effectuée avec les informations d'identification utilisateur racine ou AWS Identity and Access Management (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 requête a été effectuée par un autre service AWS.

Pour plus d'informations, consultez la section [Élément userIdentity CloudTrail](#).

Comprendre les entrées du fichier journal d'AWS Device Farm

Un journal d'activité est une configuration qui permet d'envoyer des événements sous forme de fichiers journaux à un compartiment Simple Storage Service (Amazon S3) que vous spécifiez. Les fichiers journaux CloudTrail contiennent une ou plusieurs entrées de journal. Un événement représente une demande unique provenant de n'importe quelle source et comprend des informations sur l'action demandée, la date et l'heure de l'action, les paramètres de la requête, etc. Les fichiers journaux CloudTrail ne constituent pas une série ordonnée retraçant les appels d'API publics. Ils ne suivent aucun ordre précis.

L'exemple suivant montre un CloudTrail entrée de journal illustrant le parc de périphériques `ListRuns` action :

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

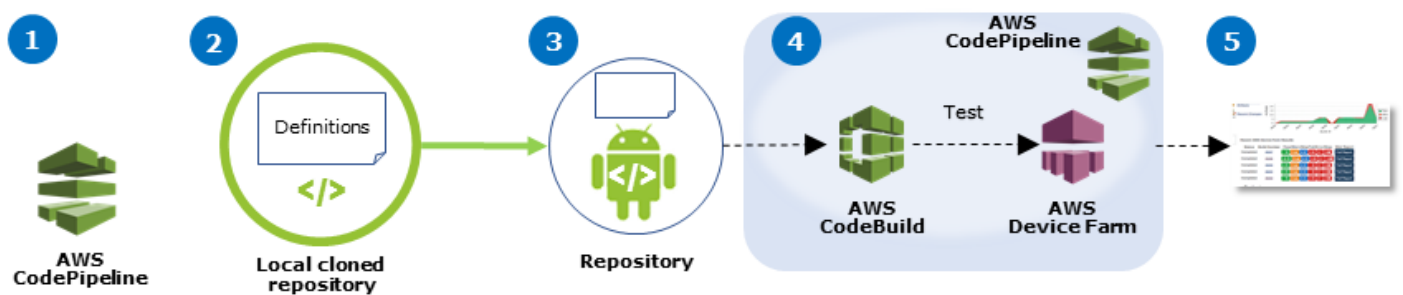
```
        "type": "BUILTIN_FUZZ",
        "status": "RUNNING",
        "totalJobs": 3,
        "platform": "ANDROID_APP",
        "result": "PENDING"
    },
    ... additional entries ...
]
}
}
}
]
```


Utilisation d'AWS Device Farm dans un CodePipeline phase de test

Vous pouvez 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 [Tutoriel : créer et tester une application Android lorsqu'elle est poussée vers GitHub](#).

Workflow to Set Up Android Application Test



| 1. Configuration | 2. Ajout de définitions | 3. Push | 4. Construire et tester | 5. Rapport |
|---------------------------------------|---|--|---|--------------------------------|
| Configurer les ressources du pipeline | Ajouter des définitions de génération et de test au package | Transmettre en mode push un package au référentiel | La génération de l'application et le test de l'artefact de sortie de génération sont lancés automatiquement | Afficher les résultats du test |

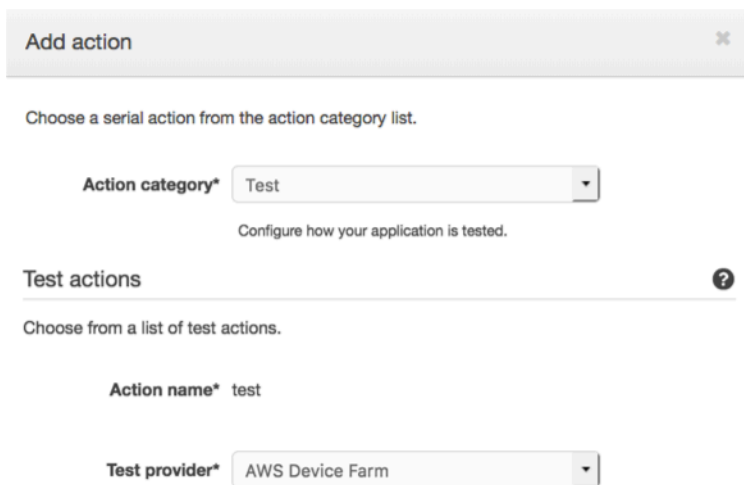
Pour savoir comment configurer un pipeline qui teste en continu une application compilée (par exemple, un fichier iOS .ipa ou un fichier Android .apk) en tant que source, consultez [Didacticiel : Test d'une application iOS à chaque chargement d'un fichier .ipa vers un compartiment Amazon S3](#).

Configurez CodePipeline pour utiliser vos tests Device Farm

Au cours de 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 parc de périphériques, effectuez un test en tant que CodePipeline action de test

1. Connectez-vous au AWS Management Console et ouvrez le CodePipeline console à <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 is a title bar 'Add action' with a close button. Below it, the instruction 'Choose a serial action from the action category list.' is displayed. The 'Action category*' dropdown menu is set to 'Test'. Below this, the text 'Configure how your application is tested.' is shown. The 'Test actions' section is expanded, showing the instruction 'Choose from a list of test actions.' The 'Action name*' field contains the text 'test'. The 'Test provider*' dropdown menu is set to 'AWS Device Farm'.

7. Dans Nom du projet, choisissez votre projet Device Farm existant ou choisissez Création d'un nouveau projet.

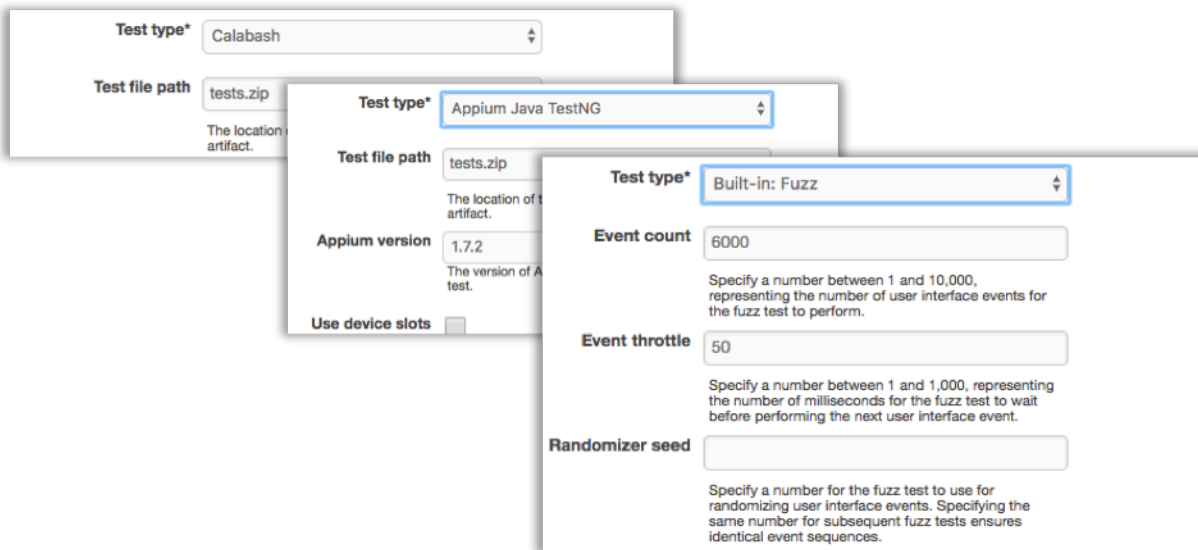
8. Dans Groupe d'appareils, choisissez votre groupe d'appareils existant, ou bien Créer un nouveau groupe d'appareils. Si vous créez un groupe d'appareils, vous devez sélectionner un ensemble d'appareils de test.
9. Dans Type d'application, choisissez la plateforme correspondant à votre application.

Device Farm Test

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

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

10. 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 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.



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.

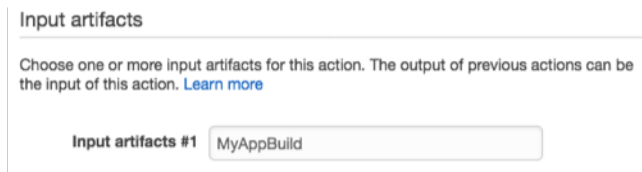
Set Radio Stats

Bluetooth **GPS**
NFC **Wifi**

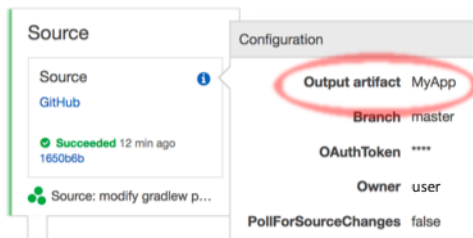
Enable app performance data capture **Enable video recording**

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.



Pour afficher le nom de l'artefact de sortie pour chaque étape dans la console CodePipeline, survolez l'icône d'information sur le schéma du pipeline. Si votre pipeline teste votre application directement à partir duLa sourceétape, choisissezMyApp. Si votre pipeline comprend unConstruireétape, choisissezMyAppBuild.



15. Au bas du panneau, choisissez Ajouter une action.
16. Dans le volet CodePipeline, choisissez Enregistrer les modifications du pipeline, puis 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 les commandes Device Farm, consultez le [AWS CLI Référence pour AWS Device Farm](#).

Pour des informations générales sur AWS CLI, consultez le [AWS Command Line Interface Guide de l'utilisateur](#) et le [AWS CLI Référence de commande](#).

FenêtresPowerShellréférence pour AWS Device Farm

Pour utiliser WindowsPowerShellpour exécuter les commandes Device Farm, consultez le[Référence de l'applet de commande Device Farm](#) dans le[AWS Tools for Windows PowerShellRéférence de l'applet de commande](#). Pour plus d'informations, voir[Configuration des outils AWS pour WindowsPowerShell](#) dans leAWS Tools for Windows PowerShellGuide de 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 kit SDK AWS et l'AWS CLI permettent d'y parvenir.

Le SDK AWS permet d'accéder à tous les services AWS, y compris Device Farm, Amazon S3, etc. Pour plus d'informations, veuillez consulter la rubrique

- les [outils et les kits SDK AWS](#)
- le [Référence d'API AWS Device Farm](#)

Exemple : utilisation du SDK AWS pour démarrer une ferme de périphériques, exécuter et collecter des artefacts

L'exemple suivant fournit une démonstration du début à la fin de la manière dont vous pouvez utiliser le SDK AWS compatible 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

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.

Rubriques

- [Résolution des problèmes liés aux tests d'applications Android dans AWS Device Farm](#)
- [Résolution des problèmes liés aux tests Appium Java JUnit dans AWS Device Farm](#)
- [Résolution des problèmes liés aux tests d'applications Web Appium Java JUnit 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](#)
- [Résolution des problèmes liés aux tests XCTest dans AWS Device Farm](#)
- [Résolution des problèmes liés aux tests d'interface utilisateur XCTest 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 plus d'informations, veuillez consulter [Utilisation des 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'`unaapt 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 plus d'informations, veuillez consulter [Utilisation des 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 plus d'informations, veuillez consulter [Utilisation des 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'`unaapt 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 plus d'informations, veuillez consulter [Utilisation des 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 pu trouver le code valide `AndroidManifest.xml` 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 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 plus d'informations, veuillez consulter [Utilisation des 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 plus d'informations, veuillez consulter [Utilisation des 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 réalisé par Device Farm, votre application utilise peut-être `AndroidFLAG_SECURE` fonctionnalité. Ce drapeau (tel que décrit dans [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 consulter les pages non bloquées par le `FLAG_SECURE` drapeau).

Résolution des problèmes liés aux tests Appium Java JUnit dans AWS Device Farm

La rubrique suivante répertorie les messages d'erreur qui s'affichent pendant le chargement des tests d'Appium Java JUnit et recommande des solutions permettant de résoudre chaque erreur.

Note

Les instructions ci-dessous sont basées sur Linux x86_64 et Mac.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_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 package Appium Java JUnit valide doit générer une sortie similaire à la suivante :

```
.
|- 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 package Appium Java JUnit est valide, vous trouverez le répertoire *dependency-jars* 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 package Appium Java JUnit est valide, vous trouverez au moins un fichier *jar* dans le répertoire *dependency-jars* :

```
.  
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything  
built from the ./src/main directory)  
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing  
everything built from the ./src/test directory)  
|- zip-with-dependencies.zip (this .zip file contains all of the items)
```

```
`- dependency-jars (this is the directory that contains all of your dependencies,
  built as JAR files)
  |- com.some-dependency.bar-4.1.jar
  |- com.another-dependency.thing-1.0.jar
  |- joda-time-2.7.jar
  `- log4j-1.2.14.jar
```

Pour plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 package Appium Java JUnit est valide, vous en trouverez au moins un *bo*ca*l* 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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

Impossible de trouver une valeur de version JUnit. Décompressez votre package de test, puis ouvrez le répertoire `dependency-jars`, vérifiez que le fichier JAR JUnit 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 package Appium Java JUnit est valide, vous trouverez le fichier de dépendance JUnit similaire au fichier jar *junit-4.10.jar* de notre exemple. Le nom doit comporter le mot clé *junit* et son numéro de version, qui est 4.10 dans cet exemple.

Pour plus d'informations, veuillez consulter [Travailler avec Appium et AWS Device Farm](#).

APPIUM_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

La version de JUnit que vous utilisez est antérieure à la version 4.10 minimale que nous prenons en charge. Modifiez la version de JUnit 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 dépendance JUnit comme *junit-4.10.jar* dans notre exemple et son numéro de version, qui est ici 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 peuvent ne pas s'exécuter correctement si la version de JUnit spécifiée dans votre package de test est antérieure à la version 4.10 minimale prise en charge.

Pour plus d'informations, veuillez consulter [Travailler avec Appium et AWS Device Farm](#).

Résolution des problèmes liés aux tests d'applications Web Appium Java JUnit 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 JUnit et recommande des solutions permettant de résoudre chaque erreur. Pour plus d'informations sur l'utilisation d'Appium avec Device Farm, voir [the section called "Appium"](#).

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 package Appium Java JUnit valide doit générer une sortie similaire à la suivante :

```
.
|- 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 package Appium Java JUnit est valide, vous trouverez le répertoire *dependency-jars* 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 package Appium Java JUnit est valide, vous trouverez au moins un fichier *jar* dans le répertoire *dependency-jars* :

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `- log4j-1.2.14.jar
```

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

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 package Appium Java JUnit est valide, vous en trouverez au moins un *bocal* 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

Impossible de trouver une valeur de version JUnit. Décompressez votre package de test, puis ouvrez le répertoire `dependency-jars`, vérifiez que le fichier JAR JUnit 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 package Appium Java JUnit est valide, vous trouverez le fichier de dépendance JUnit similaire au fichier jar *junit-4.10.jar* de notre exemple. Le nom doit comporter le mot clé *junit* et son numéro de version, qui est 4.10 dans cet exemple.

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

La version de JUnit que vous utilisez est antérieure à la version 4.10 minimale que nous prenons en charge. Modifiez la version de JUnit 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 dépendance JUnit comme *junit-4.10.jar* dans notre exemple et son numéro de version, qui est ici 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 peuvent ne pas s'exécuter correctement si la version de JUnit spécifiée dans votre package de test est antérieure à la version 4.10 minimale prise en charge.

Pour plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 package Appium Java JUnit valide doit générer une sortie similaire à la suivante :

```
.
|- 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 package Appium Java JUnit est valide, vous trouverez le répertoire *dependency-jars* 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS Device Farm](#).

APPIUM_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_

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 package Appium Java JUnit est valide, vous trouverez au moins un fichier *jar* dans le répertoire *dependency-jars*.

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `- log4j-1.2.14.jar
```

Pour plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 package Appium Java JUnit est valide, vous en trouverez au moins un *bo*ca*l* 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 package Appium Java JUnit valide doit générer une sortie similaire à la suivante :

```
.
|- 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 package Appium Java JUnit est valide, vous trouverez le répertoire *dependency-jars* 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 package Appium Java JUnit est valide, vous trouverez au moins un fichier *jar* dans le répertoire *dependency-jars*.

```
.
|- acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
|- acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
|- zip-with-dependencies.zip (this .zip file contains all of the items)
`- dependency-jars (this is the directory that contains all of your dependencies,
    built as JAR files)
    |- com.some-dependency.bar-4.1.jar
    |- com.another-dependency.thing-1.0.jar
    |- joda-time-2.7.jar
    `- log4j-1.2.14.jar
```

Pour plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 package Appium Java JUnit est valide, vous en trouverez au moins un *boçal* 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 dépendant `.whl` comme les fichiers mis en surbrillance dans le répertoire `wheelhouse`.

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|   |-- test_unittest.py
|-- wheelhouse (directory)
|   |-- Appium_Python_Client-0.20-cp27-none-any.whl
|   |-- py-1.4.31-py2.py3-none-any.whl
|   |-- pytest-2.9.0-py2.py3-none-any.whl
|   |-- selenium-2.52.0-cp27-none-any.whl
|   |-- wheel-0.26.0-py2.py3-none-any.whl
```

Pour plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 dépendant `.whl` comme les fichiers mis en surbrillance dans le répertoire `wheelhouse`. Le nom du fichier peut être différent, mais il doit se terminer par `-any.whl` ou `-linux_x86_64.whl` qui indique la plateforme utilisée. 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 Appium Python est valide, vous trouverez le répertoire `tests` 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 Appium Python est valide, vous trouverez le répertoire `tests` 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 Appium Python est valide, vous trouverez le fichier `requirements.txt` 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 numéro de `.whl` fichiers 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 `.whl` le nombre de fichiers dépendants est inférieur au nombre de lignes non vides de votre `requirements.txt` fichier, alors vous devez vous assurer de ce qui suit :

- Il y a un `.whl` fichier dépendant correspondant à chaque ligne du `requirements.txt` fichier.
- Il n'y a aucune autre ligne dans le `requirements.txt` fichier contenant des informations autres que les noms des packages de dépendances.

- Aucun nom de dépendance n'est dupliqué sur plusieurs lignes dans le `requirements.txt` fichier tel que deux lignes du fichier puissent correspondre à une `.whl` fichier dépendant.

AWS Device Farm ne prend pas en charge les lignes du `requirements.txt` les fichiers qui ne correspondent pas directement aux packages de dépendances, tels que les lignes qui spécifient les options globales pour `pip install` commande. Voir [Format de fichier des exigences](#) pour une liste d'options globales.

Pour plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 dépendant *.whl* comme les fichiers mis en surbrillance dans le répertoire *wheelhouse*.

```
.
|-- requirements.txt
|-- test_bundle.zip
|-- tests (directory)
|   |-- test_unittest.py
|-- wheelhouse (directory)
|   |-- Appium_Python_Client-0.20-cp27-none-any.whl
|   |-- py-1.4.31-py2.py3-none-any.whl
|   |-- pytest-2.9.0-py2.py3-none-any.whl
|   |-- selenium-2.52.0-cp27-none-any.whl
|   |-- wheel-0.26.0-py2.py3-none-any.whl
```

Pour plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 dépendant *.whl* comme les fichiers mis en surbrillance dans le répertoire *wheelhouse*. Le nom du fichier peut être différent, mais il doit se terminer par *-any.whl* ou *-linux_x86_64.whl* qui indique la plateforme utilisée. 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 Appium Python est valide, vous trouverez le répertoire *tests* 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 Appium Python est valide, vous trouverez le répertoire `tests` 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 Appium Python est valide, vous trouverez le fichier `requirements.txt` 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 fichier `requirements.txt` 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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 plus d'informations, veuillez consulter [Travailler avec Appium et AWS 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.

INSTRUMENTATION_TEST_PACKAGE_UNZIP_FAILED

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

⚠ Warning

Impossible d'ouvrir votre fichier APK 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 `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 plus d'informations, veuillez consulter [Utilisation d'instruments 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.

⚠ Warning

Impossible d'extraire des informations sur votre package de test. Vérifiez que le package de test est valide en exécutant la commande « `aapt debug badging <chemin de votre package de test>` », puis réessayez une fois que cette commande ne génère aucune erreur.

Au cours du processus de validation du téléchargement, Device Farm analyse les informations issues de la sortie `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 plus d'informations, veuillez consulter [Utilisation d'instruments 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.

Warning

Nous n'avons pas pu trouver la valeur du circuit d'instrumentation dans `AndroidManifest.xml`. Vérifiez que le package de test est valide en exécutant la commande « `aapt dump xmltree <path to your test package>AndroidManifest.xml` », et réessayez après avoir trouvé la valeur du moteur d'instrumentation derrière le mot clé « `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 plus d'informations, veuillez consulter [Utilisation d'instruments 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.

Warning

Nous n'avons pas pu trouver le code valideAndroidManifest.xml dans votre package de test. Vérifiez que le package de test est valide en exécutant la commande « `aapt dump xmltree` » `<path to your test package>AndroidManifest.xml` », et réessayez une fois que la commande n'affiche aucune erreur.

Au cours du 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 plus d'informations, veuillez consulter [Utilisation d'instruments 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.

Warning

Impossible de trouver le nom du package à l'intérieur de votre package de test. Vérifiez que le package de test est valide en exécutant la commande « `aapt debug badging <chemin`

de votre package de test> » et réessayez après avoir trouvé la valeur du nom du package derrière le mot clé « package: name ».

Au cours du 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 plus d'informations, veuillez consulter [Utilisation d'instruments 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 `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 .
```

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 plus d'informations, veuillez consulter [Utilisation des 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 `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 .
```

Si le package de l'application iOS est valide, vous trouverez le répertoire *Payload* dans le répertoire de travail.

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

Pour plus d'informations, veuillez consulter [Utilisation des 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 `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 .
```

Si le package de l'application iOS est valide, vous trouverez un `.app` répertoire comme `AWSDeviceFarmiOSReferenceApp.app` dans notre exemple à l'intérieur du `Charge utile` annuaire.

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

Pour plus d'informations, veuillez consulter [Utilisation des 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 `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 .
```

Si le package de l'application iOS est valide, vous trouverez le *Info.plist* fichier à l'intérieur du *.app* répertoire comme *AWSDeviceFarmiOSReferenceApp.app* dans notre exemple.

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

Pour plus d'informations, veuillez consulter [Utilisation des 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 « UIRequiredDeviceCapabilities » est spécifié, puis 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 à l'intérieur d'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 plus d'informations, veuillez consulter [Utilisation des 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 « CFBundleSupportedPlatforms» est spécifié, puis 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 à l'intérieur d'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 plus d'informations, veuillez consulter [Utilisation des 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 « CFBundleSupportedPlatforms» ne contient pas le mot clé « simulateur », puis 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 à l'intérieur d'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 plus d'informations, veuillez consulter [Utilisation des 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 « UIDeviceFamily » est spécifié, puis 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 à l'intérieur d'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 plus d'informations, veuillez consulter [Utilisation des 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 « CFBundleIdentifier » est spécifiée, puis 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 à l'intérieur d'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 plus d'informations, veuillez consulter [Utilisation des 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 « CFBundleExecutable» est spécifié, puis réessayez.

Dans l'exemple suivant, le nom du package estAWSDeviceFarmiOSReferenceApp.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 à l'intérieur d'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 plus d'informations, veuillez consulter [Utilisation des tests iOS dans AWS Device Farm](#).

Résolution des problèmes liés aux tests XCTest dans AWS Device Farm

La rubrique suivante répertorie les messages d'erreur qui s'affichent pendant le chargement des tests XCTest et recommande des solutions permettant de 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 package XCTest valide doit générer une sortie similaire à la suivante :

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

Pour plus d'informations, veuillez consulter [Utilisation de XCTest pour iOS et AWS Device Farm](#).

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 package XCTest valide, vous trouverez un répertoire avec un nom similaire à `swiftExampleTests.xctest` dans le répertoire de travail. Le nom doit se terminer par `.xctest`.

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

Pour plus d'informations, veuillez consulter [Utilisation de XCTest pour iOS et AWS Device Farm](#).

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 package XCTest est valide, vous trouverez le fichier *Info.plist* dans le répertoire *.xctest*. Dans notre exemple ci-dessous, le répertoire s'appelle *swiftExampleTests.xctest*.

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

Pour plus d'informations, veuillez consulter [Utilisation de XCTest pour iOS et AWS Device Farm](#).

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 « CFBundleIdentifier » est spécifié, 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 à l'intérieur d'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'application XCTest valide doit générer une sortie similaire à la suivante :

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

Pour plus d'informations, veuillez consulter [Utilisation de XCTest pour iOS et AWS Device Farm](#).

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 « CFBundleExecutable » est spécifié, 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 à l'intérieur d'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'application XCTest valide doit générer une sortie similaire à la suivante :

```
swiftExampleTests
```

Pour plus d'informations, veuillez consulter [Utilisation de XCTest pour iOS et AWS Device Farm](#).

Résolution des problèmes liés aux tests d'interface utilisateur XCTest dans AWS Device Farm

La rubrique suivante répertorie les messages d'erreur qui s'affichent pendant le chargement des tests XCTest UI et recommande des solutions permettant de 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.

Warning

Impossible d'ouvrir votre fichier IPA 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 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 plus d'informations, veuillez consulter [XCTest UI](#).

XCTEST_UI_TEST_PACKAGE_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 package. Décompressez votre package de test, vérifiez que le répertoire Payload s'y trouve, puis réessayez.

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 XCTest UI est valide, vous trouverez le répertoire *Payload* 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)
        |-- (any other files)
```

Pour plus d'informations, veuillez consulter [XCTest UI](#).

XCTEST_UI_TEST_PACKAGE_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 package de test, 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 `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 XCTest UI est valide, vous trouverez un répertoire `.app`, comme `swift-sampleUITests-Runner.app` dans notre exemple, dans le répertoire `Payload`.

```
.
|-- Payload (directory)
    |-- swift-sampleUITests-Runner.app (directory)
        |-- Info.plist
        |-- Plugins (directory)
            |-- swift-sampleUITests.xctest (directory)
                |-- Info.plist
                |-- (any other files)
            |-- (any other files)
```

Pour plus d'informations, veuillez consulter [XCTest UI](#).

XCTEST_UI_TEST_PACKAGE_PLUGINS_DIR_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver le répertoire Plugins à l'intérieur de votre répertoire `.app`. Décompressez votre package de test, puis ouvrez le répertoire `.app`, vérifiez que le répertoire Plugins est à l'intérieur du répertoire, puis réessayez.

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 XCTest UI est valide, vous trouverez le répertoire *Plugins* dans le répertoire *.app*. Dans notre exemple, le répertoire est appelé *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 plus d'informations, veuillez consulter [XCTest UI](#).

XCTEST_UI_TEST_PACKAGE_XCTEST_DIR_MISSING_IN_PLUGINS_DIR

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 répertoire `Plugins`. Décompressez votre package de test, puis ouvrez le répertoire `Plugins`, vérifiez que le répertoire `.xctest` est à l'intérieur du répertoire, puis réessayez.

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 XCTest UI est valide, vous trouverez un répertoire `.xctest` dans le répertoire `Plugins`. Dans notre exemple, le répertoire est appelé `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 plus d'informations, veuillez consulter [XCTest UI](#).

XCTEST_UI_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 `.app`. Décompressez votre package de test, puis ouvrez le répertoire `.app`, vérifiez que le fichier Info.plist se trouve dans le répertoire, puis réessayez.

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 XCTest UI est valide, vous trouverez le fichier *Info.plist* dans le répertoire *.app*. Dans notre exemple ci-dessous, le répertoire est appelé *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 plus d'informations, veuillez consulter [XCTest UI](#).

XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING_IN_XCTEST_DIR

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 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 XCTest UI est valide, vous trouverez le fichier *Info.plist* dans le répertoire *.xctest*. Dans notre exemple ci-dessous, le répertoire est appelé *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 plus d'informations, veuillez consulter [XCTest UI](#).

XCTEST_UI_TEST_PACKAGE_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 package de test, puis ouvrez le fichier Info.plist dans le répertoire .app, vérifiez que la clé « UIRequiredDeviceCapabilities» est spécifié, puis réessayez.

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 fichier *Info.plist* dans un répertoire *.app* 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 XCTest UI valide doit générer une sortie similaire à la suivante :

```
['armv7']
```

Pour plus d'informations, veuillez consulter [XCTest UI](#).

XCTEST_UI_TEST_PACKAGE_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 de Info.plist. Décompressez votre package de test, puis ouvrez le fichier Info.plist dans le répertoire .app, vérifiez que la clé « CFBundleSupportedPlatforms » est spécifié, puis réessayez.

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 fichier *Info.plist* dans un répertoire *.app* 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 XCTest UI valide doit générer une sortie similaire à la suivante :

```
['iPhoneOS']
```

Pour plus d'informations, veuillez consulter [XCTest UI](#).

XCTEST_UI_TEST_PACKAGE_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 package de test, puis ouvrez le fichier Info.plist dans le répertoire .app, vérifiez que la valeur de la clé « CFBundleSupportedPlatforms» ne contient pas le mot clé « simulateur », puis réessayez.

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 fichier *Info.plist* dans un répertoire *.app* 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 XCTest UI valide doit générer une sortie similaire à la suivante :

```
['iPhoneOS']
```

Si le package XCTest UI est valide, la valeur ne doit pas contenir le mot clé `simulator`.

Pour plus d'informations, veuillez consulter [XCTest UI](#).

XCTEST_UI_TEST_PACKAGE_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 de Info.plist. Décompressez votre package de test, puis ouvrez le fichier Info.plist dans le répertoire .app, vérifiez que la clé « UIDeviceFamily » est spécifié, puis réessayez.

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 fichier *Info.plist* dans un répertoire *.app* 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 XCTest UI valide doit générer une sortie similaire à la suivante :

```
[1, 2]
```

Pour plus d'informations, veuillez consulter [XCTest UI](#).

XCTEST_UI_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 dans le répertoire .app, vérifiez que la clé « CFBundleIdentifier» est spécifié, puis réessayez.

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 fichier *Info.plist* dans un répertoire *.app* 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 XCTest UI valide doit générer une sortie similaire à la suivante :

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

Pour plus d'informations, veuillez consulter [XCTest UI](#).

XCTEST_UI_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 dans le répertoire .app, vérifiez que la clé « CFBundleExecutable » est spécifiée, puis réessayez.

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 fichier *Info.plist* dans un répertoire *.app* 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 XCTest UI valide doit générer une sortie similaire à la suivante :

```
XCTRunner
```

Pour plus d'informations, veuillez consulter [XCTest UI](#).

XCTEST_UI_TEST_PACKAGE_TEST_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 dans le répertoire .xctest. Décompressez votre package de test, puis ouvrez le fichier Info.plist dans le répertoire .xctest, vérifiez que la clé « CFBundleIdentifier » est spécifié, puis réessayez.

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 fichier *Info.plist* dans un répertoire *.app* 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 XCTest UI valide doit générer une sortie similaire à la suivante :

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

Pour plus d'informations, veuillez consulter [XCTest UI](#).

XCTEST_UI_TEST_PACKAGE_TEST_EXECUTABLE_VALUE_MISSING

Si le message suivant s'affiche, procédez comme suit pour corriger le problème.

Warning

Impossible de trouver la valeur exécutable à l'intérieur du fichier Info.plist dans le répertoire .xctest. Décompressez votre package de test, puis ouvrez le fichier Info.plist dans le répertoire .xctest, vérifiez que la clé « CFBundleExecutable » est spécifiée, puis réessayez.

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 fichier *Info.plist* dans un répertoire *.app* 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 XCTest UI valide doit générer une sortie similaire à la suivante :

```
swift-sampleUITests
```

Pour plus d'informations, veuillez consulter [XCTest UI](#).

Sécurité dans AWS Device Farm

Chez AWS, la sécurité dans le cloud est notre priorité numéro 1. En tant que client AWS, vous bénéficiez d'un centre de données et d'une architecture réseau conçus pour répondre aux exigences des organisations les plus pointilleuses en termes de sécurité.

La sécurité est une responsabilité partagée entre AWS et vous. Le [modèle de responsabilité partagée](#) décrit cela comme la sécurité du cloud et la sécurité dans le cloud :

- Sécurité du cloud : AWS est responsable de la protection de l'infrastructure qui exécute des services AWS dans le cloud AWS. AWS vous fournit également les services que vous pouvez utiliser en toute sécurité. Des auditeurs tiers testent et vérifient régulièrement l'efficacité de notre sécurité dans le cadre des [AWS programmes de conformité](#). Pour en savoir plus sur les programmes de conformité qui s'appliquent à AWS Device Farm, veuillez consulter [Services AWS concernés par le 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 atteindre 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 la 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](#)
- [Enregistrement et surveillance dans Device Farm](#)
- [Bonnes pratiques de sécurité pour Device Farm](#)

Gestion des identités et des accès dans AWS Device Farm

Public ciblé

La façon dont vous utilisez AWS Identity and Access Management (IAM) varie en fonction du travail que vous effectuez dans Device Farm.

Utilisateur du service : si vous utilisez le service Device Farm pour effectuer votre travail, votre administrateur vous fournit les informations d'identification et les autorisations dont vous avez besoin. Au fur et à mesure que vous utilisez de plus en plus de fonctionnalités de Device Farm dans le cadre de votre travail, il se peut que vous ayez besoin d'autorisations supplémentaires. En comprenant bien la gestion des accès, vous saurez demander les autorisations appropriées à votre administrateur. Si vous ne parvenez pas à accéder à une fonctionnalité dans Device Farm, consultez [Résolution des problèmes d'identité et d'accès à AWS Device Farm](#).

Administrateur du service — Si vous êtes responsable des ressources de Device Farm au sein de votre entreprise, vous avez probablement un accès complet à Device Farm. C'est à vous de déterminer les fonctionnalités et les ressources de Device Farm auxquelles les utilisateurs de votre service doivent accéder. Vous devez ensuite soumettre les demandes à votre administrateur IAM pour modifier les autorisations des utilisateurs de votre service. Consultez les informations sur cette page pour comprendre les concepts de base d'IAM. Pour en savoir plus sur la manière dont votre entreprise peut utiliser IAM avec Device Farm, consultez [Comment AWS Device Farm fonctionne avec IAM](#).

Administrateur IAM : si vous êtes administrateur IAM, vous souhaitez peut-être en savoir plus sur la manière dont vous pouvez rédiger des politiques pour gérer l'accès à Device Farm. Pour consulter des exemples de politiques basées sur l'identité Device Farm que vous pouvez utiliser dans IAM, consultez [Exemples de politiques basées sur l'identité d'AWS Device Farm](#)

Authentification par des identités

L'authentification correspond au processus par lequel vous vous connectez à AWS avec vos informations d'identification. Vous devez vous authentifier (être connecté à AWS) en tant qu'utilisateur racine d'un compte AWS, en tant qu'utilisateur IAM ou en endossant un rôle IAM.

Vous pouvez vous connecter à AWS en tant qu'identité fédérée à l'aide des informations d'identification fournies par le biais d'une source d'identité. AWS IAM Identity Center Les utilisateurs (IAM Identity Center), l'authentification de connexion unique de votre entreprise et vos informations d'identification Google ou Facebook sont des exemples d'identités fédérées. Lorsque vous vous

connectez avec une identité fédérée, votre administrateur aura précédemment configuré une fédération d'identités avec des rôles IAM. Lorsque vous accédez à AWS en utilisant la fédération, vous endossez indirectement un rôle.

Selon le type d'utilisateur que vous êtes, vous pouvez vous connecter à la AWS Management Console ou au portail d'accès AWS. Pour plus d'informations sur la connexion à AWS, consultez [Connexion à votre Compte AWS](#) dans le Guide de l'utilisateur Connexion à AWS.

Si vous accédez à AWS par programmation, AWS fournit un kit de développement logiciel (SDK) et une interface de ligne de commande (CLI) pour signer cryptographiquement vos demandes en utilisant vos informations d'identification. Si vous n'utilisez pas les outils AWS, vous devez signer les requêtes vous-même. Pour plus d'informations sur l'utilisation de la méthode recommandée pour signer des demandes vous-même, consultez [Signature des demandes d'API AWS](#) dans le Guide de l'utilisateur IAM.

Quelle que soit la méthode d'authentification que vous utilisez, vous devrez peut-être fournir des informations de sécurité supplémentaires. Par exemple, AWS vous recommande d'utiliser l'authentification multifactorielle (MFA) pour améliorer la sécurité de votre compte. Pour en savoir plus, veuillez consulter [Authentification multifactorielle](#) dans le Guide de l'utilisateur AWS IAM Identity Center et [Utilisation de l'authentification multifactorielle \(MFA\) dans l'interface AWS](#) dans le Guide de l'utilisateur IAM.

Utilisateur root Compte AWS

Lorsque vous créez un Compte AWS, vous commencez avec une seule identité de connexion disposant d'un accès complet à tous les Services AWS et ressources du compte. Cette identité est appelée utilisateur root du Compte AWS. Vous pouvez y accéder en vous connectant à l'aide de l'adresse électronique et du mot de passe que vous avez utilisés pour créer le compte. Il est vivement recommandé de ne pas utiliser l'utilisateur root pour vos tâches quotidiennes. Protégez vos informations d'identification d'utilisateur root et utilisez-les pour effectuer les tâches que seul l'utilisateur root peut effectuer. Pour obtenir la liste complète des tâches qui vous imposent de vous connecter en tant qu'utilisateur root, consultez [Tâches nécessitant les informations d'identification de l'utilisateur root](#) dans le Guide de l'utilisateur IAM.

Utilisateurs et groupes IAM

Un [utilisateur IAM](#) est une identité dans votre Compte AWS qui dispose d'autorisations spécifiques pour une seule personne ou application. Dans la mesure du possible, nous vous recommandons de vous appuyer sur des informations d'identification temporaires plutôt que de créer des utilisateurs

IAM ayant des informations d'identification à long terme tels que les clés d'accès. Toutefois, si certains cas d'utilisation spécifiques nécessitent des informations d'identification à long terme avec les utilisateurs IAM, nous vous recommandons de faire pivoter les clés d'accès. Pour plus d'informations, consultez [Rotation régulière des clés d'accès pour les cas d'utilisation nécessitant des informations d'identification](#) dans le Guide de l'utilisateur IAM.

Un [groupe IAM](#) est une identité qui concerne un ensemble d'utilisateurs IAM. Vous ne pouvez pas vous connecter en tant que groupe. Vous pouvez utiliser les groupes pour spécifier des autorisations pour plusieurs utilisateurs à la fois. Les groupes permettent de gérer plus facilement les autorisations pour de grands ensembles d'utilisateurs. Par exemple, vous pouvez avoir un groupe nommé IAMAdmins et accorder à ce groupe les autorisations d'administrer des ressources IAM.

Les utilisateurs sont différents des rôles. Un utilisateur est associé de manière unique à une personne ou une application, alors qu'un rôle est conçu pour être endossé par tout utilisateur qui en a besoin. Les utilisateurs disposent d'informations d'identification permanentes, mais les rôles fournissent des informations d'identification temporaires. Pour en savoir plus, consultez [Quand créer un utilisateur IAM \(au lieu d'un rôle\)](#) dans le Guide de l'utilisateur IAM.

Rôles IAM

Un [rôle IAM](#) est une entité au sein de votre Compte AWS qui dispose d'autorisations spécifiques. Le concept ressemble à celui d'utilisateur IAM, mais le rôle IAM n'est pas associé à une personne en particulier. Vous pouvez temporairement endosser un rôle IAM dans la AWS Management Console en [changeant de rôle](#). Vous pouvez obtenir un rôle en appelant une opération d'API AWS CLI ou AWS à l'aide d'une URL personnalisée. Pour plus d'informations sur les méthodes d'utilisation des rôles, consultez [Utilisation de rôles IAM](#) dans le Guide de l'utilisateur IAM.

Les rôles IAM avec des informations d'identification temporaires sont utiles dans les cas suivants :

- Accès utilisateur fédéré – Pour attribuer des autorisations à une identité fédérée, vous créez un rôle et définissez des autorisations pour le rôle. Quand une identité externe s'authentifie, l'identité est associée au rôle et reçoit les autorisations qui sont définies par celui-ci. Pour obtenir des informations sur les rôles pour la fédération, consultez [Création d'un rôle pour un fournisseur d'identité tiers \(fédération\)](#) dans le Guide de l'utilisateur IAM. Si vous utilisez IAM Identity Center, vous configurez un jeu d'autorisations. IAM Identity Center met en corrélation le jeu d'autorisations avec un rôle dans IAM afin de contrôler à quoi vos identités peuvent accéder après leur authentification. Pour plus d'informations sur les jeux d'autorisations, consultez [Jeux d'autorisations](#) dans le Guide de l'utilisateur AWS IAM Identity Center.

- Autorisations d'utilisateur IAM temporaires : un rôle ou un utilisateur IAM peut endosser un rôle IAM pour profiter temporairement d'autorisations différentes pour une tâche spécifique.
- Accès intercompte : vous pouvez utiliser un rôle IAM pour permettre à un utilisateur (principal de confiance) d'un compte différent d'accéder aux ressources de votre compte. Les rôles constituent le principal moyen d'accorder l'accès intercompte. Toutefois, certains Services AWS vous permettent d'attacher une politique directement à une ressource (au lieu d'utiliser un rôle en tant que proxy). Pour en savoir plus sur la différence entre les rôles et les politiques basées sur les ressources pour l'accès intercompte, consultez [Différence entre les rôles IAM et les politiques basées sur les ressources](#) dans le Guide de l'utilisateur IAM.
- Accès interservices : certains Services AWS utilisent des fonctions dans d'autres Services AWS. Par exemple, lorsque vous effectuez un appel dans un service, il est courant que ce service exécute des applications dans Amazon EC2 ou stocke des objets dans Amazon S3. Un service peut le faire en utilisant les autorisations d'appel du principal, une fonction du service ou un rôle lié au service.
 - Forward access sessions (FAS) – Lorsque vous utilisez un utilisateur ou un rôle IAM pour effectuer des actions dans AWS, vous êtes considéré comme un principal. Lorsque vous utilisez certains services, vous pouvez effectuer une action qui déclenche une autre action dans un autre service. FAS utilise les autorisations du principal appelant Service AWS, combinées à la demande Service AWS pour effectuer des demandes aux services en aval. Les demandes FAS ne sont formulées que lorsqu'un service reçoit une demande qui, pour aboutir, a besoin d'interagir avec d'autres ressources ou Services AWS. Dans ce cas, vous devez disposer d'autorisations nécessaires pour effectuer les deux actions. Pour plus de détails sur la politique relative à la transmission de demandes FAS, consultez [Sessions de transmission d'accès](#).
- Fonction du service : il s'agit d'un [rôle IAM](#) attribué à un service afin de réaliser des actions en votre nom. Un administrateur IAM peut créer, modifier et supprimer une fonction de service à partir d'IAM. Pour plus d'informations, consultez [Création d'un rôle pour la délégation d'autorisations à un Service AWS](#) dans le Guide de l'utilisateur IAM.
- Rôle lié au service : un rôle lié au service est un type de fonction de service lié à un Service AWS. Le service peut endosser le rôle afin d'effectuer une action en votre nom. Les rôles liés à un service s'affichent dans votre Compte AWS et sont détenus par le service. Un administrateur IAM peut consulter, mais ne peut pas modifier, les autorisations concernant les rôles liés à un service.
- Applications s'exécutant sur Amazon EC2 : vous pouvez utiliser un rôle IAM pour gérer des informations d'identification temporaires pour les applications s'exécutant sur une instance EC2 et effectuant des demandes d'API AWS CLI ou AWS. Cette solution est préférable au stockage

des clés d'accès au sein de l'instance EC2. Pour attribuer un rôle AWS à une instance EC2 et le rendre disponible à toutes les applications associées, vous pouvez créer un profil d'instance attaché à l'instance. Un profil d'instance contient le rôle et permet aux programmes qui s'exécutent sur l'instance EC2 d'obtenir des informations d'identification temporaires. Pour plus d'informations, consultez [Utilisation d'un rôle IAM pour accorder des autorisations à des applications s'exécutant sur des instances Amazon EC2](#) dans le Guide de l'utilisateur IAM.

Pour savoir dans quel cas utiliser des rôles ou des utilisateurs IAM, consultez [Quand créer un rôle IAM \(au lieu d'un utilisateur\)](#) 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, veuillez consulter [Références des éléments de politique JSON IAM](#) dans le Guide de l'utilisateur IAM.

Actions

Les administrateurs peuvent utiliser les politiques JSON AWS 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. Les actions de politique possèdent généralement le même nom que l'opération d'API AWS associée. Il existe quelques exceptions, telles que les actions avec autorisations uniquement qui n'ont pas d'opération API correspondante. Certaines opérations nécessitent également plusieurs actions dans une politique. Ces actions supplémentaires sont nommées actions dépendantes.

Intégration d'actions dans une stratégie 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 JSON AWS 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 pour lesquels l'action s'applique. Les instructions doivent inclure un élément `Resource` ou `NotResource`. Il est recommandé de définir

une ressource à l'aide de son [Amazon Resource Name \(ARN\)](#). Vous pouvez le faire pour des actions qui prennent en charge un type de ressource spécifique, connu sous la dénomination autorisations de niveau ressource.

Pour les actions qui ne sont pas compatibles avec les autorisations de niveau ressource, telles que les opérations de liste, 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 des ARN, consultez [Noms ARN \(Amazon Resource Name\) et Espaces de noms du service AWS](#).

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 leurs ARN par des virgules.

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

```
"resource2"
```

Pour consulter la liste des types de ressources Device Farm et de leurs ARN, consultez la section [Types de ressources définis par AWS Device Farm](#) dans le 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 JSON AWS 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 (ou le bloc Condition) vous permet de spécifier des conditions lorsqu'une instruction est appliquée. L'élément Condition est facultatif. 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.

Si vous spécifiez plusieurs éléments Condition dans une instruction, ou plusieurs clés dans un seul élément Condition, AWS les évalue à l'aide d'une opération AND logique. Si vous spécifiez plusieurs valeurs pour une seule clé de condition, AWS évalue la condition à l'aide d'une opération OR logique. Toutes les conditions doivent être remplies avant que les autorisations associées à l'instruction ne soient accordées.

Vous pouvez aussi utiliser des variables d'espace réservé quand vous spécifiez des conditions. Par exemple, vous pouvez accorder à un utilisateur IAM l'autorisation d'accéder à une ressource uniquement si elle est balisée avec son nom d'utilisateur IAM. Pour plus d'informations, consultez [Éléments des politiques IAM : variables et balises](#) dans le Guide de l'utilisateur IAM.

AWS prend en charge les clés de condition globales et les clés de condition spécifiques à un service. Pour afficher toutes les clés de condition globales AWS, consultez [Clés de contexte de condition globale AWS](#) 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 afficher toutes les clés de condition globales AWS, veuillez consulter la rubrique [Clés de contexte de condition globale AWS](#) 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 (ACL).

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 balises, vous devez fournir les informations de balise 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 STSAPI telles que [AssumeRole](#) ou [GetFederationToken](#).

Rôles liés à un service

Les [rôles liés à un service](#) permettent aux services AWS d'accéder à des ressources dans 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.

Fonctions du service

Device Farm ne prend pas en charge les rôles de service.

Cette fonction permet à un service d'endosser une [fonction du 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 fonctions du 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 des accès à l'aide de politiques

Vous contrôlez les accès dans AWS en créant des politiques et en les attachant à des identités AWS ou à des ressources. Une politique est un objet dans AWS qui, lorsqu'il est associé à une identité ou à une ressource, définit les autorisations de ces dernières. AWS évalue ces politiques lorsqu'un principal (utilisateur, utilisateur racine ou séance de rôle) envoie une demande. Les autorisations dans les politiques déterminent si la demande est autorisée ou refusée. La plupart des politiques sont stockées dans AWS en tant que documents JSON. Pour plus d'informations sur la structure et le contenu des déclarations de politique JSON, consultez [Présentation des politiques JSON](#) dans le Guide de l'utilisateur IAM.

Les administrateurs peuvent utiliser les politiques JSON AWS pour spécifier qui a accès à quoi. C'est-à-dire, 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. Pour octroyer aux utilisateurs des autorisations d'effectuer des actions sur les ressources dont ils ont besoin, un administrateur IAM peut créer des politiques IAM. L'administrateur peut ensuite ajouter les politiques IAM aux rôles et les utilisateurs peuvent endosser les rôles.

Les politiques IAM définissent les autorisations d'une action, quelle que soit la méthode que vous utilisez pour exécuter l'opération. Par exemple, supposons que vous disposiez d'une politique qui autorise l'action `iam:GetRole`. Un utilisateur avec cette politique peut obtenir des informations utilisateur à partir de la AWS Management Console, de la AWS CLI ou de l'API AWS.

Politiques basées sur l'identité

Les politiques basées sur l'identité sont des déclarations de politique d'autorisations JSON que vous pouvez attacher à une identité telle qu'un utilisateur, un groupe d'utilisateurs ou un rôle IAM. Ces politiques contrôlent quel type d'actions des utilisateurs et des rôles peuvent exécuter, sur quelles ressources et dans quelles conditions. Pour découvrir comment créer une politique basée sur l'identité, consultez [Création de politiques IAM](#) dans le Guide de l'utilisateur IAM.

Les politiques basées sur l'identité peuvent être classées comme des politiques en ligne ou des politiques gérées. Les politiques en ligne sont intégrées directement à un utilisateur, groupe ou rôle. Les politiques gérées sont des politiques autonomes que vous pouvez attacher à plusieurs utilisateurs, groupes et rôles dans votre Compte AWS. Les politiques gérées incluent les politiques gérées par AWS et les politiques gérées par le client. Pour découvrir comment choisir entre une politique gérée et une politique 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.

| Modification | 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 d'autres types de politiques moins courantes. Ces types de politiques peuvent définir le nombre maximum d'autorisations qui vous sont accordées par des types de politiques plus courants.

- **Limite d'autorisations** : une limite d'autorisations est une fonction avancée dans laquelle vous définissez le nombre maximal d'autorisations qu'une politique basée sur l'identité peut accorder à une entité IAM (utilisateur ou rôle IAM). Vous pouvez définir une limite d'autorisations pour une entité. Les autorisations qui en résultent représentent la combinaison des politiques basées sur l'identité d'une entité et de ses limites d'autorisation. Les politiques basées sur les ressources qui spécifient l'utilisateur ou le rôle dans le champ `Principal` ne sont pas limitées par les limites d'autorisations. Un refus explicite dans l'une de ces politiques remplace l'autorisation. Pour plus d'informations sur les limites d'autorisations, consultez [Limites d'autorisations pour des entités IAM](#) dans le Guide de l'utilisateur IAM.
- **Politiques de contrôle des services (SCP)** - les SCP sont des politiques JSON qui spécifient le nombre maximal d'autorisations pour une organisation ou une unité d'organisation (OU) dans AWS Organizations. AWS Organizations est un service qui vous permet de regrouper et de gérer de façon centralisée plusieurs Comptes AWS détenus par votre entreprise. Si vous activez toutes les fonctions d'une organisation, vous pouvez appliquer les politiques de contrôle des services (SCP) à l'un ou à l'ensemble de vos comptes. La SCP limite les autorisations pour les entités dans les comptes membres, y compris dans chaque Utilisateur racine d'un compte AWS. Pour plus d'informations sur les organisations et les SCP, consultez [Fonctionnement des SCP](#) dans le Guide de l'utilisateur AWS Organizations.
- **politiques de séance** : les politiques de séance sont des politiques avancées que vous utilisez en tant que paramètre lorsque vous créez par programmation une séance temporaire pour un rôle ou un utilisateur fédéré. Les autorisations de la séance obtenue sont une combinaison des politiques basées sur l'identité de l'utilisateur ou du rôle et des politiques de séance. Les autorisations peuvent également provenir d'une politique basée sur les ressources. Un refus explicite dans l'une de ces politiques remplace l'autorisation. Pour plus d'informations, consultez [Politiques de séance](#) dans le Guide de l'utilisateur IAM.

Plusieurs types de politique

Lorsque plusieurs types de politiques s'appliquent à la requête, les autorisations obtenues sont plus compliquées à comprendre. Pour découvrir la façon dont AWS détermine s'il convient d'autoriser une demande en présence de plusieurs types de politiques, veuillez consulter [Logique d'évaluation de 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 exécuter des tâches à l'aide de AWS Management

Console, AWS CLI ou de l'API AWS. 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 :

- Démarrer avec AWS gérées et évoluez vers les autorisations de moindre privilège - Pour commencer à accorder des autorisations à vos utilisateurs et charges de travail, utilisez les politiques gérées AWS qui accordent des autorisations dans de nombreux cas d'utilisation courants. Elles sont disponibles dans votre Compte AWS. Nous vous recommandons de réduire encore les autorisations en définissant des politiques gérées par le client AWS qui sont spécifiques à vos cas d'utilisation. Pour de plus amples informations, consultez [AWS Politiques gérées](#) ou [AWS Politiques gérées pour les activités professionnelles](#) dans le Guide de l'utilisateur IAM.
- Accorder 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.

- Utiliser 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 via un Service AWS spécifique, comme AWS CloudFormation. Pour plus d'informations, consultez [Conditions pour éléments de politique JSON IAM](#) dans le Guide de l'utilisateur IAM.
- Utilisez IAM Access Analyzer pour valider vos politiques IAM afin de garantir des autorisations sécurisées et fonctionnelles - IAM Access Analyzer 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 politique IAM Access Analyzer](#) dans le Guide de l'utilisateur IAM.
- Authentification multifactorielle (MFA) nécessaire : si vous avez un scénario qui nécessite des utilisateurs IAM ou un utilisateur root dans votre Compte AWS, activez l'authentification multifactorielle pour une sécurité renforcée. Pour exiger le MFA lorsque des opérations d'API sont appelées, ajoutez des conditions MFA à vos politiques. Pour plus d'informations, consultez [Configuration de l'accès aux API protégé par 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 nécessaires pour réaliser cette action sur la console ou par programmation à l'aide de l'AWS CLI ou de l'API AWS.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
```



```

        "iam:ListGroupsWithUser",
        "iam:ListAttachedUserPolicies",
        "iam:ListUserPolicies",
        "iam:GetUser"
    ],
    "Resource": ["arn:aws:iam::*:user/${aws:username}"]
},
{
    "Sid": "NavigateInConsole",
    "Effect": "Allow",
    "Action": [
        "iam:GetGroupPolicy",
        "iam:GetPolicyVersion",
        "iam:GetPolicy",
        "iam:ListAttachedGroupPolicies",
        "iam:ListGroupPolicies",
        "iam:ListPolicyVersions",
        "iam:ListPolicies",
        "iam:ListUsers"
    ],
    "Resource": "*"
}
]
}

```

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`.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GetTestGridProject",

```

```

    "Effect": "Allow",
    "Action": [
      "devicefarm:GetTestGridProject"
    ],
    "Resource": "arn:aws:devicefarm:us-west-2:111122223333:testgrid-
project:123e4567-e89b-12d3-a456-426655441111"
  },
  {
    "Sid": "ViewProjectInfo",
    "Effect": "Allow",
    "Action": [
      "devicefarm:ListTestGridSessions",
      "devicefarm:ListTestGridSessionActions",
      "devicefarm:ListTestGridSessionArtifacts"
    ],
    "Resource": "arn:aws:devicefarm:us-west-2:111122223333:testgrid-*:123e4567-
e89b-12d3-a456-426655441111/*"
  }
]
}

```

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 :

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "$id": "scheduleTestRuns",
      "effect": "Allow",
      "Action": [ "devicefarm:CreateUpload", "devicefarm:ScheduleRun" ],
      "Resource": [
        "arn:aws:devicefarm:us-west-2:111122223333:project:123e4567-e89b-12d3-
a456-426655440000",
        "arn:aws:devicefarm:us-west-2:111122223333:*:123e4567-e89b-12d3-
a456-426655440000/*",
      ]
    }
  ]
}

```

```

]
}

```

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.

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ListTestGridProjectSessions",
      "Effect": "Allow",
      "Action": [
        "devicefarm:ListTestGridSession*",
        "devicefarm:GetTestGridSession",
        "devicefarm:ListTestGridProjects"
      ],
      "Resource": [
        "arn:aws:devicefarm:us-west-2:testgrid-project:*/*"
        "arn:aws:devicefarm:us-west-2:testgrid-session:*/*"
      ],
      "Condition": {
        "StringEquals": {"aws:TagKey/Owner": "${aws:username}"}
      }
    }
  ]
}

```

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 dans la AWS Management Console selon lequel vous n'êtes pas autorisé à exécuter 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:GetRun` 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'action `iam:PassRole`, vos politiques doivent être mises à jour pour vous permettre de transmettre un rôle à Device Farm.

Certains Services AWS vous permettent de transmettre un rôle existant à ce service, au lieu de créer une nouvelle fonction du service ou rôle lié à un service. Pour ce faire, un utilisateur doit 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 une fonction du service. Mary ne dispose pas des autorisations nécessaires pour transférer 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 encore besoin d'aide, contactez votre administrateur AWS. 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](#). En effet, vous lui accorderiez ainsi 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 créer une entité IAM (utilisateur ou rôle) pour la personne ou l'application qui a besoin d'y accéder. Ils utiliseront les informations d'identification de cette entité pour accéder à AWS. Vous devez ensuite associer une politique à l'entité qui lui accorde les autorisations appropriées dans Device Farm.

Pour démarrer immédiatement, consultez [Création de votre premier groupe et utilisateur délégué 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 (ACL), vous pouvez utiliser ces politiques pour donner l'accès à vos ressources.

Pour en savoir plus, 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 octroyer l'accès à vos ressources à des Comptes AWS dont vous êtes propriétaire, consultez la section [Fournir l'accès à un utilisateur IAM dans un autre Compte AWS que vous possédez](#) dans le Guide de l'utilisateur IAM.
- Pour savoir comment octroyer l'accès à vos ressources à des tiers Comptes AWS, consultez [Fournir l'accès aux Comptes AWS appartenant à 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 découvrir quelle est la différence entre l'utilisation des rôles et l'utilisation des politiques basées sur les ressources pour l'accès entre comptes, consultez [Différence entre les rôles IAM et les politiques basées sur les ressources](#) dans le Guide de l'utilisateur IAM.

Validation de la conformité pour AWS Device Farm

Les auditeurs tiers évaluent la sécurité et la conformité de AWS Device Farm dans le cadre de plusieurs programmes de conformité AWS. Il s'agit notamment des certifications SOC, PCI, FedRAMP, HIPAA parmi d'autres. AWS Device Farm n'est concerné par aucun programme de conformité AWS.

Pour obtenir la liste des services AWS relevant de programmes de conformité spécifiques, consultez les [Services AWS relevant de programmes de conformité](#). Pour obtenir des renseignements généraux, consultez [Programmes de conformité AWS](#).

Vous pouvez télécharger les rapports de l'audit externe avec AWS Artifact. Pour de plus amples informations, veuillez consulter [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 de Quick Start sur la sécurité et la conformité](#) – Ces guides de déploiement traitent de considérations architecturales et fournissent des étapes pour déployer des environnements de base axés sur la sécurité et la conformité dans AWS.
- [Ressources de conformité AWS](#) : cet ensemble de manuels et de guides peut s'appliquer à votre secteur d'activité et à votre emplacement.
- [Évaluer les ressources à l'aide de règles](#) dans le [AWS Config Guide du 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](#) : ce service AWS fournit une vue complète de votre état de sécurité au sein d'AWS qui vous permet de vérifier votre conformité aux normes du secteur et aux bonnes pratiques de 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 responsable de la protection de l'infrastructure globale sur laquelle l'ensemble du AWS Cloud s'exécute. 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 en savoir plus 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 [AWS et RGPD \(Règlement général sur la protection des données\)](#) sur le AWSBlog de sécurité.

À des fins de protection des données, nous vous recommandons de protéger les informations d'identification Compte AWS et de configurer les comptes utilisateur individuels avec AWS IAM Identity Center ou AWS Identity and Access Management (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.
- Utilisez les certificats SSL/TLS pour communiquer avec les ressources AWS. Nous exigeons TLS 1.2 et recommandons TLS 1.3.
- Configurez une API (Interface de programmation) et le journal de l'activité des utilisateurs avec AWS CloudTrail.
- Utilisez des solutions de chiffrement AWS, ainsi que tous les contrôles de sécurité par défaut au sein des 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 FIPS (Federal Information Processing Standard) 140-2 lorsque vous accédez à AWS via une CLI (Interface de ligne de commande) ou une API (Interface de programmation), utilisez un point de terminaison FIPS (Federal Information Processing Standard). Pour en savoir plus sur les points de terminaison FIPS (Federal Information Processing Standard) disponibles, consultez [Federal Information Processing Standard \(FIPS\) 140-2](#) (Normes de traitement de l'information fédérale).

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 Name (Nom). Cela inclut lorsque vous travaillez avec Device Farm ou une autre entreprise Services AWS à l'aide de la console, de l'API ou AWS des SDK. AWS CLI Toutes les données que vous saisissez 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 les demandes HTTPS (SSL/TLS) signées, sauf indication contraire. Tout le contenu extrait ou placé dans Amazon S3 via des URL de téléchargement est chiffré à l'aide du protocole SSL/TLS. Pour plus d'informations sur la façon dont les requêtes HTTPS sont signées dans AWS, veuillez consulter [Signature des demandes d'API AWS](#) dans Références générales AWS.

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, mais toutes les métadonnées (ARN, dates de téléchargement, noms de fichiers, etc.) sont conservées pour une utilisation future. Le tableau suivant contient la période de conservation pour différents types de contenu.

| Type de contenu | Période de conservation (jours) |
|---|---------------------------------|
| Applications chargées | 30 |
| Paquets de test chargés | 30 |
| Journaux | 400 |
| Enregistrements vidéo et autres artefacts | 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 plus d'informations, consultez [Utilisation d'appareils privés dans AWS Device Farm](#).

Gestion des clés

Device Farm ne propose actuellement aucune gestion de clé externe pour le chiffrement des données, 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 instances Amazon EC2 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 réseau AWS n'est pas garantie, et il vous incombe de sécuriser les connexions Internet de vos applications.

Résilience dans AWS Device Farm

L'infrastructure mondiale AWS s'articule autour de régions et de zones de disponibilité AWS. Les Régions fournissent plusieurs zones de disponibilité physiquement séparées et isolées, reliées par un réseau à latence faible, 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 les régions et les zones de disponibilité AWS, consultez [AWS Infrastructure mondiale](#).

Device Farm étant disponible dans leus-west-2 Région uniquement, 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 est protégé par les procédures de sécurité du réseau mondial AWS. Pour plus d'informations sur les services de sécurité AWS et la manière dont AWS protège l'infrastructure, consultez la section [Sécurité du cloud AWS](#). Pour concevoir votre environnement AWS en utilisant les meilleures pratiques en matière de sécurité de l'infrastructure, consultez la section [Protection de l'infrastructure](#) dans le Security Pillar AWS Well-Architected Framework (Pilier de sécurité de l'infrastructure Well-Architected Framework).

Vous utilisez AWS appels d'API 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 nous 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 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 Sélénium ne peuvent pas communiquer entre elle sans une partie tierce intermédiaire, extérieure à AWS.

Tout le trafic vers SeleniumWebDriver les contrôleurs doivent être créés via le point de terminaison HTTPS généré avec `createTestGridUrl`.

La fonctionnalité de test du navigateur de bureau ne prend pas en charge la configuration des points de terminaison Amazon VPC pour le moment. Il vous incombe de vous assurer que chaque instance de test Device Farm dispose d'un accès sécurisé aux ressources qu'elle teste.

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 est à jour, de par sa conception, permettant l'utilisation de 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 par rapport au [vulnérabilité dans 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 périphériques privés sont maintenus conformément à votre accord avec AWS.

Device Farm fait tout son possible pour empêcher les applications des clients de commettre des actions telles que `rooting` ou `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, voir [Pratiques de sécurité](#) sur le [RubyGem](#) 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 le [Détection des failles de sécurité](#) sur GitHub.
- Pour plus d'informations sur le vérificateur de dépendances Maven de l'Open Web Application Security Project (OWASP), voir [GUÊPE DependencyCheck](#) sur le site Web de l'OWASP.

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 AWS est informé d'un cas où les données du client, notamment les résultats des tests ou les fichiers écrits sur un périphérique public, sont accessibles à un autre client, AWS contacte les clients concernés conformément aux alertes d'incident standard et aux stratégies de signalement utilisées dans l'ensemble des services AWS.

Enregistrement et surveillance dans Device Farm

Ce service prend en charge AWS CloudTrail, qui est un service qui enregistre AWS appels pour votre Compte AWS et envoie les fichiers journaux dans un compartiment Amazon S3. En utilisant les informations collectées par CloudTrail, vous pouvez déterminer quelles demandes ont été envoyées avec succès à Services AWS, qui a fait la demande, quand elle a été faite, etc. Pour en savoir

plus sur [CloudTrail](#), y compris comment l'activer et trouver vos fichiers journaux, consultez le [AWS CloudTrail Guide de l'utilisateur](#).

Pour plus d'informations sur l'utilisation [CloudTrail](#) avec Device Farm, voir [Journalisation des appels d'API AWS Device Farm avec AWS CloudTrail](#).

Bonnes pratiques 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 [Utilisation d'environnements de test personnalisés](#)

Limites dans AWS Device Farm

La liste suivante décrit les limites actuelles d'AWS Device Farm :

- La taille maximale d'un fichier d'application que vous pouvez charger est de 4 Go.
- 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 chiffre peut être augmenté sur demande.)
- Il n'y a pas de limite au nombre d'exécutions que vous pouvez planifier.
- La durée d'une session d'accès à distance est limitée à 150 minutes.
- La durée d'une exécution de test automatisée 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'existe pas de limite au nombre d'appareils que vous pouvez inclure dans une série de tests. Le nombre d'appareils, ou de tâches, sur lesquels vous pouvez exécuter des 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 limitée sur AWS Device Farm est de 5. Vous pouvez demander une augmentation de ce nombre 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.

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 sur 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 d'AWS Device Farm au plugin Jenkins CI](#)
- [Plug-in AWS Device Farm Gradle](#)

Intégration d'AWS Device Farm au plugin Jenkins CI

Ce plugin 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 de [Étape 1 : Installation du plugin](#).

Cette section contient une série de procédures pour configurer et utiliser le plug-in Jenkins CI avec AWS Device Farm.

Rubriques

- [Étape 1 : Installation du plugin](#)
- [Étape 2 : Création d'un AWS Identity and Access Management utilisateur pour votre plugin Jenkins CI](#)
- [Étape 3 : Instructions pour la toute première configuration](#)
- [Étape 4 : Utilisation du plugin dans une tâche Jenkins](#)
- [Dépendances](#)

Les images suivantes montrent les fonctions du plug-in Jenkins CI.

The screenshot shows the Jenkins CI interface for a project named "Hello World App". On the left, there is a navigation menu with options: "Back to Dashboard", "Status", "Changes", "Workspace", "Build Now", "Delete Project", "Configure", and "AWS Device Farm". The main content area is titled "Project Hello World App" and includes links for "Workspace" and "Recent Changes". Below these are two main sections: "Build History" and "Recent AWS Device Farm Results".

Build History

| Build Number | Timestamp |
|--------------|-----------------------|
| #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 :



Jenkins > Hello World App > #19

Back to Project
Status
Changes
Console Output
Edit Build Information
Delete Build
AWS Device Farm
Previous Build

Artifacts of Hello World App #19

 [AWS Device Farm Results](#) /

- [Amazon Kindle Fire HDX 7 \(WiFi\)](#)
- [Motorola DROID Ultra \(Verizon\)](#)
- [Samsung Galaxy Note 4 \(AT&T\)](#)
- [Samsung Galaxy S5 \(AT&T\)](#)
- [Samsung Galaxy Tab 4 10.1 Nook \(WiFi\)](#)

 [\(all files in zip\)](#)

Étape 1 : Installation du plugin

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 AWS Identity and Access Management utilisateur pour votre plugin Jenkins CI

Nous vous recommandons de ne pas utiliser votre AWS compte root pour accéder à Device Farm. Créez plutôt un nouveau AWS Identity and Access Management utilisateur (IAM) (ou utilisez un utilisateur IAM existant) dans votre AWS, 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.

Autoriser l'utilisateur IAM à 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 AWS le compte 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. Pour Description, saisissez une description qui vous aidera à associer cet utilisateur IAM à votre projet Jenkins.
7. Pour Document de stratégie, saisissez l'instruction suivante :

```
{
  "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 : Instructions pour la toute première configuration

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'au Parc d'appareils AWS en tête.
5. Copiez vos identifiants de sécurité de votre [Étape 2 : créer un utilisateur IAM](#) et collez votre ID de clé d'accès et votre clé d'accès secrète dans leurs zones respectives.
6. Choisissez Save (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 Ajouter une action après la construction et sélectionnez Exécuter des tests sur 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 Save (Enregistrer).

Dépendances

Le plug-in Jenkins CI nécessite le kit SDK AWS Mobile 1.10.5 ou une version ultérieure. Pour plus d'informations et pour installer le kit SDK, consultez [Kit SDK AWS Mobile](#).

Plug-in AWS Device Farm Gradle

Ce plugin permet l'intégration d'AWS Device Farm au système de génération Gradle dans Android Studio. Pour plus d'informations, consultez [Gradle](#).

Note

Pour télécharger le plugin Gradle, rendez-vous sur [GitHub](#) et suivez les instructions figurant 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 plug-in Device Farm Gradle.

Rubriques

- [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](#)
- [Étape 4 : Configuration des types de tests](#)
- [Dépendances](#)

Étape 1 : Création du plugin AWS Device Farm Gradle

Ce plugin permet l'intégration d'AWS Device Farm au système de génération 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. Allez à [GitHub](#) et clonez le dépôt.
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](#)

Étape 3 : Génération d'un utilisateur IAM

AWS Identity and Access Management(IAM) vous aide à gérer les autorisations et les politiques relatives à l'utilisation deAWSressources. Cette rubrique explique comment générer 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 votreAWScompte root pour accéder à Device Farm. Créez plutôt un nouvel utilisateur IAM (ou utilisez un utilisateur IAM existant) dans votreAWS, puis accédez à Device Farm avec cet utilisateur IAM.

Note

LeAWSle compte 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. Sélectionnez Create (Créer).
6. Choisissez Télécharger les informations d'identification et enregistrez-les à un emplacement où vous pourrez facilement les récupérer ultérieurement.
7. Choisissez Close (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. ChoisissezCliquez icioù 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.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "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, voir [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 [Utilisation d'instruments 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 [Utilisation des types de tests 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 JUnit et 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 plus d'informations, consultez [Intégré : Fuzz \(Android et iOS\)](#).

Instrumentation

Device Farm fournit un support pour l'instrumentation (JUnit, Espresso, Robotium ou tout autre test basé sur l'instrumentation) pour Android. Pour plus d'informations, consultez [Utilisation d'instruments 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  
}
```

Dépendances

Exécution

- Le plugin Device Farm Gradle nécessite AWS Mobile SDK 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

Historique du document

Le tableau suivant décrit les modifications importantes apportées à la documentation depuis la dernière version de ce manuel.

| Modification | Description | Date de modification |
|---|---|----------------------|
| Support AL2 | Device Farm prend désormais en charge l'environnement de test AL2 pour Android. En savoir plus sur AL2 . | 6 novembre 2023 |
| Migration d'environnements de test standard vers des environnements de test personnalisés | Guide de migration mis à jour pour documenter la dépréciation des documents pour les tests en mode standard en décembre 2023. | 3 septembre 2023 |
| 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 : <ul style="list-style-type: none">• Appium (Python)• Appium (Python) | 20 avril 2020 |
| Nouvelles informations sur la sécurité et informations sur | Pour faciliter la sécurisation des services AWS et la rendre plus complète, une nouvelle section sur la sécurité a été | 27 mars 2020 |

| Modification | Description | Date de modification |
|--|--|----------------------|
| le balisage des ressources AWS. | <p>créée. Pour en savoir plus, consultez Sécurité dans AWS Device Farm</p> <p>Une nouvelle section sur le balisage dans Device Farm a été ajoutée. Pour de plus amples informations sur le balisage, veuillez consulter Marquage dans Device Farm.</p> | |
| Suppression de l'accès direct aux appareils. | <p>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.</p> | 9 septembre 2019 |
| Mise à jour de la configuration du plug-in Gradle | <p>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.</p> | 16 août 2019 |
| Nouvelle exigence pour les exécutions de test avec XCTest | <p>Pour les tests utilisant le framework XCTest, Device Farm nécessite désormais un package d'application conçu pour les tests. En savoir plus sur the section called "XCTest".</p> | 4 février 2019 |
| Prise en charge des types de test Appium Node.js et Ruby dans des environnements personnalisés | <p>Vous pouvez désormais exécuter vos tests dans les environnements de test personnalisés Appium Node.js et Appium Ruby. En savoir plus sur Utilisation des types de tests dans AWS Device Farm.</p> | 10 janvier 2019 |

| Modification | Description | Date de modification |
|---|---|----------------------|
| 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 Utilisation des types de tests dans AWS Device Farm . | 2 octobre 2018 |
| 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 Utilisation d'environnements de test personnalisés . | 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 le cadre de 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 Utilisation d'AWS Device Farm dans un CodePipeline phase de test . | 19 juillet 2018 |

| Modification | Description | Date de modification |
|--|--|----------------------|
| 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 Utilisation d'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 Définissez le délai d'exécution pour les tests dans AWS Device Farm . | 9 février 2017 |
| Mise en forme réseau | Vous pouvez désormais simuler des connexions et des conditions réseau pour une série de tests. En savoir plus sur Simulez les connexions réseau et les conditions d'exécution 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 Utilisation de l'accès à distance . | 19 avril 2016 |

| Modification | Description | Date de modification |
|---|---|----------------------|
| Libre-service pour les emplacements d'appareils | Vous pouvez désormais acheter des emplacements d'appareils à l'aide d'AWS Management Console, de l'AWS Command Line Interface ou de l'API. Découvrez comment Achetez un emplacement pour appareil dans Device Farm. | 22 mars 2016 |
| Comment arrêter les exécutions de test | Vous pouvez désormais arrêter les exécutions de test à l'aide d'AWS Management Console, de l'AWS Command Line Interface ou de l'API. Découvrez comment Arrêter une exécution dans AWS Device Farm. | 22 mars 2016 |
| Nouveaux types de tests XCTest UI | Vous pouvez désormais exécuter des tests personnalisés XCTest UI sur des applications iOS. En savoir plus sur les types de tests XCTest UI. | 8 mars 2016 |
| 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 Utilisation des types de tests 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 Utilisation des 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 les tests d'appareils iOS et l'exécution de tests iOS (y compris XCTest) dans Utilisation des types de tests dans AWS Device Farm. | 4 août 2015 |

| Modification | Description | Date de modification |
|---------------------------|--|----------------------|
| Première version publique | Il s'agit de la première version publique du manuel AWS Device Farm Developer Guide. | 13 juillet 2015 |

Glossaire AWS

Pour connaître la terminologie la plus récente d'AWS, consultez le [Glossaire AWS](#) dans la Référence Glossaire AWS.

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.