Guia do Desenvolvedor

AWS Device Farm

Versão da API 2015-06-23

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

AWS Device Farm: Guia do Desenvolvedor

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

As marcas comerciais e imagens de marcas da Amazon não podem ser usadas no contexto de nenhum produto ou serviço que não seja da Amazon, nem de qualquer maneira que possa gerar confusão entre os clientes ou que deprecie ou desprestigie a Amazon. Todas as outras marcas comerciais que não pertencem à Amazon pertencem a seus respectivos proprietários, que podem ou não ser afiliados, patrocinados pela Amazon ou ter conexão com ela.

Table of Contents

O que é o AWS Device Farm? 1
Teste automatizado de aplicações 1
Interação por acesso remoto 1
Terminologia 2
Configurar 3
Configuração 4
Etapa 1: inscrever-se em AWS 4
Etapa 2: criar ou usar um usuário do IAM em sua AWS conta 4
Etapa 3: dar permissão ao usuário do IAM para acessar o Device Farm 5
Próxima etapa 6
Conceitos básicos 7
Pré-requisitos 7
Etapa 1: Fazer login no console do 8
Etapa 2: criar um projeto 8
Etapa 3: Criar e iniciar uma execução 8
Etapa 4: visualizar os resultados da execução 10
Próximas etapas 11
Compra de slots de dispositivos 12
Comprar slots de dispositivos (console) 12
Comprar um slot de dispositivo (AWS CLI) 14
Comprar um slot de dispositivo (API) 18
Cancelando um slot de dispositivo 18
Cancelar um slot de dispositivo (console) 19
Cancelar um slot de dispositivo (AWS CLI) 19
Cancelar um slot de dispositivo (API) 19
Conceitos 20
Dispositivos 20
Dispositivos compatíveis 21
Grupos de dispositivos 21
Dispositivos privados 21
Marcas de dispositivo 21
Slots para dispositivo 21
Aplicações de dispositivos pré-instalados 22
Recursos dos dispositivos 22

Ambientes de teste 22
Ambiente de teste padrão 23
Ambiente de teste personalizado 23
Execuções 23
Configuração da execução 24
Retenção de arquivos de execução 24
Estado do dispositivo de execução 24
Execuções paralelas 25
Configurar o tempo limite de execução 25
Anúncios nas execuções 25
Mídias nas execuções 25
Tarefas comuns nas execuções 25
Apps 25
Instrumentação de aplicações 26
Nova assinatura de aplicações nas execuções 26
Aplicações ofuscadas nas execuções 26
Relatórios 26
Retenção de relatório 27
Componentes do relatório 27
Logs nos relatórios 27
Tarefas comuns relacionadas aos relatórios 27
Sessões 27
Dispositivos compatíveis com acesso remoto 28
Retenção de arquivos de sessão 28
Instrumentação de aplicações 28
Nova assinatura de aplicações nas sessões 28
Aplicações ofuscadas nas sessões 28
Projetos 29
Criação de um projeto 29
Pré-requisitos 29
Criar um projeto (console) 29
Criar um projeto (AWS CLI) 30
Criar um projeto (API) 30
Visualizando a lista de projetos 30
Pré-requisitos 31
Visualizar a lista de projetos (console) 31

Visualizar a lista de projetos (AWS CLI) ... 31
Visualizar a lista de projetos (API) ... 31
Execuções de teste ... 33
Criando uma execução de teste ... 33
Pré-requisitos ... 34
Criar uma execução de teste (console) ... 34
Criar uma execução de teste (AWS CLI) ... 37
Criar uma execução de teste (API) ... 47
Próximas etapas ... 48
Definindo o tempo limite de execução ... 48
Pré-requisitos ... 49
Definir o tempo limite de execução de um projeto ... 49
Definir o tempo limite de execução para uma execução de teste ... 49
Simulação de conexões e condições de rede ... 50
Configurar a modelagem de rede ao programar uma execução de teste ... 50
Criar um perfil de rede ... 51
Alterar as condições da rede durante o teste ... 53
Parando uma corrida ... 53
Interromper uma execução (console) ... 53
Interromper uma execução (AWS CLI) ... 55
Interromper uma execução (API) ... 57
Visualizando uma lista de execuções ... 57
Visualizar uma lista de execuções (console) ... 57
Visualizar uma lista de execuções (AWS CLI) ... 57
Visualizar uma lista de execuções (API) ... 58
Criação de um pool de dispositivos ... 58
Pré-requisitos ... 58
Criar um grupo de dispositivos (console) ... 58
Criar um grupo de dispositivos (AWS CLI) ... 60
Criar um grupo de dispositivos (API) ... 61
Analisar resultados ... 61
Visualizando relatórios de teste ... 62
Baixar artefatos ... 71
Marcação no Device Farm ... 76
Marcar recursos ... 76
Pesquisa de recursos por tag ... 77

Remover tags de recursos 78
Estruturas de teste e testes integrados 79
Testar estruturas 79
Estruturas de teste de aplicações Android 79
Estruturas de teste de aplicações iOS 79
Estruturas de teste de aplicações web 79
Estruturas em um ambiente de teste personalizado 79
Suporte da versão do Appium 79
Tipos de teste integrado 80
Appium 80
Suporte à versão 80
Integração com os testes do Appium 81
Testes do Android 96
Estruturas de teste de aplicações Android 96
Tipos de teste incorporados para Android 97
Instrumentação 97
Testes do iOS 100
Estruturas de teste de aplicações iOS 100
Tipos de teste integrados para iOS 101
XCTest 101
XCTest UI 103
Testes de aplicações Web 107
Regras para dispositivos de acesso limitado e ilimitado 107
Testes integrados 108
Integrado: Fuzz (Android e iOS) 108
Ambientes de teste personalizados 110
Sintaxe da especificação de teste 111
Exemplo da especificação de teste 113
Ambiente de teste Android 119
Software compatível 120
Intervalos de IP compatíveis com o ambiente de teste do Amazon Linux 2 122
Usando a `devicefarm-cli` ferramenta 122
Seleção de um host de teste do Android 124
Exemplo de arquivo de especificações de teste 125
Migrar para o Amazon Linux 2 Test Host 129
Variáveis de ambiente 132

Variáveis de ambiente comuns 132
Variáveis de ambiente Appium Java JUnit 134
Variáveis de ambiente do TestNG 135
XCUITest variáveis de ambiente 135
Migrar testes 136
Considerações ao migrar 136
Etapas da migração 137
Estrutura do Appium 138
Instrumentação do Android 138
Migrando os testes existentes do iOS XCUITest 138
Ampliação do modo personalizado 138
Configurando o PIN de um dispositivo 139
Acelerando os testes baseados em Appium 140
Usando webhooks e outros APIs 142
Adicionar arquivos extras ao seu pacote de teste 144
Acesso remoto 147
Criar uma sessão 147
Pré-requisitos 148
Crie uma sessão com o console do Device Farm 148
Próximas etapas 148
Usando uma sessão 149
Pré-requisitos 149
Use uma sessão no console do Device Farm 149
Próximas etapas 150
Dicas e truques 150
Recuperando resultados da sessão de sessões de acesso remoto 150
Pré-requisitos 151
Visualização de detalhes da sessão 151
Download de vídeo ou logs de sessão 151
Dispositivos privados 152
Criar um perfil da instância 153
Solicite dispositivos privados adicionais 155
Criação de uma execução de teste ou sessão de acesso remoto 157
Seleção de dispositivos privados 158
Regras de ARN do dispositivo 159
Regras de rótulos de instâncias de dispositivos 160

Regras de ARN da instância ... 160
Criar um grupo de dispositivos privados ... 161
Criação de um grupo de dispositivos privados com dispositivos privados (AWS CLI) ... 163
Criação de um pool de dispositivos privados com dispositivos privados (API) ... 164
Ignorar a nova assinatura de aplicações ... 164
Ignorar a nova assinatura da aplicação em dispositivos Android ... 166
Ignorar a nova assinatura da aplicação em dispositivos iOS ... 166
Criar uma sessão de acesso remoto para confiar na sua aplicação ... 167
Amazon VPC em todas as regiões ... 168
Visão geral do emparelhamento de VPC em diferentes regiões VPCs ... 169
Pré-requisitos para usar o Amazon VPC ... 170
Estabelecendo uma conexão de peering entre dois VPCs ... 171
Atualizando as tabelas de rotas para VPC-1 e VPC-2 ... 171
Criação de grupos-alvo ... 172
Criar um Network Load Balancer ... 174
Criar um serviço de endpoint da VPC ... 175
Criação de uma configuração de VPC endpoint no aplicativo ... 175
Criando uma execução de teste ... 176
Criação de sistemas VPC dimensionáveis ... 176
Encerramento de dispositivos privados no Device Farm ... 176
Conectividade da VPC ... 177
AWS controle de acesso e IAM ... 179
Perfis vinculados ao serviço ... 180
Permissões de perfil vinculado ao serviço para o Device Farm ... 181
Criar um perfil vinculado ao serviço para o Device Farm ... 184
Editar um perfil vinculado ao serviço para o Device Farm ... 184
Excluir um perfil vinculado ao serviço para o Device Farm ... 184
Regiões compatíveis com as funções vinculadas ao serviço Device Farm ... 185
Pré-requisitos ... 186
Conectando o Amazon VPC ... 187
Limites ... 188
Usando serviços de endpoint de VPC - Legacy ... 189
Antes de começar ... 190
Etapa 1: Criação de um Network Load Balancer ... 191
Etapa 2: criar um serviço de endpoint da VPC ... 194
Etapa 3: criar uma configuração de endpoint da VPC ... 195

Etapa 4: Criar uma execução de teste 196
Registro de chamadas de API do AWS CloudTrail com 197
Informações do AWS Device Farm em CloudTrail 197
Compreensão das entradas do arquivo de log do AWS Device Farm 198
Integração com o AWS Device Farm 201
Configure CodePipeline para usar seus testes do Device Farm 202
AWS CLI referência 206
PowerShell Referência do Windows 207
Automatização do Device Farm 208
Exemplo: usar o AWS SDK para iniciar uma execução do Device Farm e coletar artefatos 208
Solução de problemas 213
Solução de problemas de aplicativos Android 213
ANDROID_APP_UNZIP_FAILED 213
ANDROID_APP_AAPT_DEBUG_BADGING_FAILED 214
ANDROID_APP_PACKAGE_NAME_VALUE_MISSING 215
ANDROID_APP_SDK_VERSION_VALUE_MISSING 216
ANDROID_APP_AAPT_DUMP_XMLTREE_FAILED 217
ANDROID_APP_DEVICE_ADMIN_PERMISSIONS 218
Certas janelas do meu aplicativo Android mostram uma tela em branco ou preta 219
Solução de problemas do Appium Java JUnit 220
APPIUM_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED 220
APPIUM_JAVA_JUNIT_TEST_PACKAGE_DEPENDENCY_DIR_MISSING 221
APPIUM_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DIR 222
APPIUM_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING 223
APPIUM_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JAR 224
APPIUM_JAVA_JUNIT_TEST_PACKAGE_JUNIT_VERSION_VALUE_UNKNOWN 226
APPIUM_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION 227
Solução de problemas do Appium Java web JUnit 228
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED 228
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_DEPENDENCY_DIR_MISSING 229
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DIR .. 230
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING 231
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JAR 232
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_JUNIT_VERSION_VALUE_UNKNOWN ... 234
APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION 235
Solução de problemas do Appium Java TestNG 236

APPIUM_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED 236
APPIUM_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSING 237
APPIUM_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DIR 238
APPIUM_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING 239
APPIUM_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JAR 240
Solução de problemas do Appium Java TestNG web 242
APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED 242
APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSING 243
APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DIR 244
APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING 245
APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JAR 246
Solução de problemas do Appium Python 248
APPIUM_PYTHON_TEST_PACKAGE_UNZIP_FAILED 248
APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING 249
APPIUM_PYTHON_TEST_PACKAGE_INVALID_PLATFORM 250
APPIUM_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING 251
APPIUM_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME 252
APPIUM_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISSING 253
APPIUM_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION 254
APPIUM_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS_FAILED 255
APPIUM_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED 257
APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEELS_INSUFFICIENT 258
Solução de problemas do Appium Python web 259
APPIUM_WEB_PYTHON_TEST_PACKAGE_UNZIP_FAILED 260
APPIUM_WEB_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING 261
APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PLATFORM 262
APPIUM_WEB_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING 263
APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME 264
APPIUM_WEB_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISSING 265
APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION 266
APPIUM_WEB_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS_FAILED 267
APPIUM_WEB_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED 268
Solução de problemas de testes de instrumentação 270
INSTRUMENTATION_TEST_PACKAGE_UNZIP_FAILED 270
INSTRUMENTATION_TEST_PACKAGE_AAPT_DEBUG_BADGING_FAILED 271
INSTRUMENTATION_TEST_PACKAGE_INSTRUMENTATION_RUNNER_VALUE_MISSING 272

INSTRUMENTATION_TEST_PACKAGE_AAPT_DUMP_XMLTREE_FAILED 273
INSTRUMENTATION_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSING 274
Solução de problemas de aplicativos iOS 275
IOS_APP_UNZIP_FAILED 275
IOS_APP_PAYLOAD_DIR_MISSING 276
IOS_APP_APP_DIR_MISSING 277
IOS_APP_PLIST_FILE_MISSING 278
IOS_APP_CPU_ARCHITECTURE_VALUE_MISSING 279
IOS_APP_PLATFORM_VALUE_MISSING 280
IOS_APP_WRONG_PLATFORM_DEVICE_VALUE 281
IOS_APP_FORM_FACTOR_VALUE_MISSING 283
IOS_APP_PACKAGE_NAME_VALUE_MISSING 284
IOS_APP_EXECUTABLE_VALUE_MISSING 285
Solução de problemas XCTest 286
XCTEST_TEST_PACKAGE_UNZIP_FAILED 287
XCTEST_TEST_PACKAGE_XCTEST_DIR_MISSING 287
XCTEST_TEST_PACKAGE_PLIST_FILE_MISSING 288
XCTEST_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING 289
XCTEST_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING 290
Solução de problemas de XCTest interface 292
XCTEST_UI_TEST_PACKAGE_UNZIP_FAILED 292
XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_MISSING 293
XCTEST_UI_TEST_PACKAGE_APP_DIR_MISSING 294
XCTEST_UI_TEST_PACKAGE_PLUGINS_DIR_MISSING 295
XCTEST_UI_TEST_PACKAGE_XCTEST_DIR_MISSING_IN_PLUGINS_DIR 296
XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING 296
XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING_IN_XCTEST_DIR 297
XCTEST_UI_TEST_PACKAGE_CPU_ARCHITECTURE_VALUE_MISSING 298
XCTEST_UI_TEST_PACKAGE_PLATFORM_VALUE_MISSING 300
XCTEST_UI_TEST_PACKAGE_WRONG_PLATFORM_DEVICE_VALUE 301
XCTEST_UI_TEST_PACKAGE_FORM_FACTOR_VALUE_MISSING 302
XCTEST_UI_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING 304
XCTEST_UI_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING 305
XCTEST_UI_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSING 306
XCTEST_UI_TEST_PACKAGE_TEST_EXECUTABLE_VALUE_MISSING 308
XCTEST_UI_TEST_PACKAGE_MULTIPLE_APP_DIRS 309

XCTEST_UI_TEST_PACKAGE_MULTIPLE_IPA_DIRS 310
XCTEST_UI_TEST_PACKAGE_BOTH_APP_E_IPA_DIR_PRESENT 311
XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_PRESENT_IN_ZIP 312
Segurança 314
Gerenciamento de identidade e acesso 315
Público 315
Autenticar com identidades 315
Como o AWS Device Farm funciona com o IAM 319
Gerenciar o acesso usando políticas 324
Exemplos de políticas baseadas em identidade 326
Solução de problemas 331
Validação de conformidade 334
Proteção de dados 335
Criptografia em trânsito 336
Criptografia em repouso 336
Retenção de dados 336
Gerenciamento de dados 337
Gerenciamento de chaves 338
Privacidade do tráfego entre redes 338
Resiliência 338
Segurança da infraestrutura 339
Segurança da infraestrutura para teste de dispositivos físicos 339
Segurança da infraestrutura para teste de navegador de desktop 340
Análise de configuração e vulnerabilidade 341
Resposta a incidentes 342
Registro em log e monitoramento 342
Práticas recomendadas de segurança 342
Limites 344
Ferramentas e plug-ins 346
Plug-in Jenkins CI 346
Dependências 349
Instalando o plug-in Jenkins CI 349
Criação de um usuário do IAM para seu plug-in Jenkins CI 350
Configurando o plug-in Jenkins CI pela primeira vez 352
Usando o plug-in em um trabalho do Jenkins 352
Plug-in Gradle do Device Farm 353

Dependências 354
Criação do plug-in Gradle do Device Farm 354
Configuração do plug-in Gradle do Device Farm 355
Geração de um usuário do IAM no plug-in Device Farm Gradle 357
Configuração de tipos de teste 359
Histórico de documentos 361
AWS Glossário 366
..... ccclxvii

O que é o AWS Device Farm?

O Device Farm é um serviço de teste de aplicações que você pode usar para testar e interagir com suas aplicações Android, iOS e web em telefones e tablets reais e físicos hospedados pela Amazon Web Services (AWS).

Existem duas maneiras principais de usar o Device Farm:

- Testes automatizados de aplicativos usando uma variedade de estruturas de teste.
- Acesso remoto aos dispositivos nos quais você pode carregar, executar e interagir com aplicativos em tempo real.

> **Note**
>
> O Device Farm está disponível somente na região `us-west-2` (Oregon).

Teste automatizado de aplicações

O Device Farm permite que você carregue seus próprios testes ou use testes de compatibilidade incorporados e sem scripts. Como o teste é executado em paralelo, vários dispositivos começam a ser testados em questão de minutos.

À medida que os testes são concluídos, um relatório de teste que contém resultados de alto nível, registros de baixo nível, pixel-to-pixel capturas de tela e dados de desempenho é atualizado.

O Device Farm suporta testes de aplicativos Android e iOS nativos e híbridos, incluindo aqueles criados com PhoneGap Titanium, Xamarin, Unity e outras estruturas. É compatível com acesso remoto a aplicativos Android e iOS para testes interativos. Para obter mais informações sobre tipos de teste compatíveis, consulte Estruturas de teste e testes integrados no AWS Device Farm.

Interação por acesso remoto

O acesso remoto permite que você deslize o dedo, faça gestos e interaja com um dispositivo por meio de um navegador da web em tempo real. Há várias situações em que a interação em tempo real com um dispositivo é útil. Por exemplo, os representantes de atendimento ao cliente podem

orientar os clientes em meio ao uso ou à configuração do dispositivo. Eles também podem orientar os clientes em meio ao uso de aplicativos executados em um dispositivo específico. Você pode instalar aplicativos em um dispositivo executado em uma sessão de acesso remoto e reproduzir os problemas do cliente ou os bugs relatados.

Durante uma sessão de acesso remoto, o Device Farm coleta detalhes sobre as ações que ocorrem quando você interage com o dispositivo. Os logs com esses detalhes e uma captura de vídeo da sessão são produzidos no final da sessão.

Terminologia

O Device Farm apresenta os seguintes termos que definem a forma como as informações são organizadas:

grupo de dispositivos

: Um conjunto de dispositivos que normalmente compartilham características semelhantes como plataforma, fabricante ou modelo.

trabalho

: Uma solicitação para o Device Farm para testar um único aplicativo em um único dispositivo. Um trabalho contém um ou mais pacotes.

medição

: Refere-se à cobrança dos dispositivos. Você pode ver as referências a dispositivos ou dispositivos de acesso ilimitado na documentação e na referência de API. Para obter mais informações sobre preços, consulte Definição de preço do AWS Device Farm.

project

: Um espaço de trabalho lógico que contém execuções, uma para cada teste de um único aplicativo em um ou mais dispositivos. Você pode usar projetos para organizar os espaços de trabalho da forma que você escolher. Por exemplo, você pode ter um projeto por título de aplicativo ou um projeto por plataforma. Você pode criar quantos projetos necessitar.

relatório

: Contém informações sobre uma execução, que é uma solicitação para que o Device Farm teste um único aplicativo em um ou mais dispositivos. Para obter mais informações, consulte Relatórios no AWS Device Farm.

run

Uma compilação específica de seu aplicativo, com um conjunto específico de testes, para execução em um conjunto específico de dispositivos. Uma execução produz um relatório dos resultados. A execução contém um ou mais trabalhos. Para obter mais informações, consulte Execuções.

sessão

Uma interação em tempo real com um dispositivo real, físico, por meio de um navegador da web. Para obter mais informações, consulte Sessões.

pacote

A organização hierárquica de testes em um pacote de testes. Um pacote contém um ou mais testes.

teste

Um caso de teste específico em um pacote de testes.

Para obter mais informações sobre o Device Farm, consulte Conceitos.

Configurar

Para usar o Device Farm, consulte Configuração.

Configurar o AWS Device Farm

Antes de usar o Device Farm pela primeira vez, você deve concluir as seguintes tarefas:

Tópicos

- Etapa 1: inscrever-se em AWS
- Etapa 2: criar ou usar um usuário do IAM em sua AWS conta
- Etapa 3: dar permissão ao usuário do IAM para acessar o Device Farm
- Próxima etapa

Etapa 1: inscrever-se em AWS

Cadastre-se na Amazon Web Services (AWS).

Se você não tiver um Conta da AWS, conclua as etapas a seguir para criar um.

Para se inscrever em um Conta da AWS

1. Abra a https://portal.aws.amazon.com/billing/inscrição.
2. Siga as instruções online.

   Parte do procedimento de inscrição envolve receber uma chamada telefônica ou uma mensagem de texto e inserir um código de verificação pelo teclado do telefone.

   Quando você se inscreve em um Conta da AWS, um Usuário raiz da conta da AWSé criado. O usuário-raiz tem acesso a todos os Serviços da AWS e recursos na conta. Como prática recomendada de segurança, atribua o acesso administrativo a um usuário e use somente o usuário-raiz para executar tarefas que exigem acesso de usuário-raiz.

Etapa 2: criar ou usar um usuário do IAM em sua AWS conta

Recomendamos que você não use sua conta AWS root para acessar o Device Farm. Em vez disso, crie um usuário AWS Identity and Access Management (IAM) (ou use um existente) em sua AWS conta e acesse o Device Farm com esse usuário do IAM.

Para obter mais informações, consulte Creating an IAM User (AWS Management Console).

Etapa 3: dar permissão ao usuário do IAM para acessar o Device Farm

Dê ao usuário do IAM permissão para acessar o Device Farm. Para isso, crie uma política de acesso no IAM e, em seguida, atribua a política de acesso ao usuário do IAM, como indicado a seguir.

> **Note**
>
> A conta AWS raiz ou o usuário do IAM que você usa para concluir as etapas a seguir deve ter permissão para criar a seguinte política do IAM e anexá-la ao usuário do IAM. Para obter mais informações, consulte Trabalhar com políticas.

1. Crie uma política com o seguinte corpo JSON. Dê a ele um título descritivo, como*DeviceFarmAdmin*.

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

   Para obter mais informações sobre a criação de políticas do IAM, consulte Creating IAM Policies no Guia do usuário do IAM.

2. Anexe a política do IAM que você criou ao novo usuário. Para obter mais informações sobre como anexar políticas do IAM aos usuários, consulte Adding and Removing IAM Policies no Guia do usuário do IAM.

Anexar a política fornece ao usuário do IAM acesso a todas as ações e recursos do Device Farm associados a esse usuário do IAM. Para obter informações sobre como restringir os usuários do IAM a um conjunto limitado de ações e recursos do Device Farm, consulte Gerenciamento de identidade e acesso no AWS Device Farm.

Próxima etapa

Agora está tudo pronto para começar a usar o Device Farm. Consulte Conceitos básicos do Device Farm.

Conceitos básicos do Device Farm

Este passo a passo mostra como usar o Device Farm para testar um aplicativo nativo para Android ou iOS. Use o console do Device Farm para criar um projeto, carregar um arquivo .apk ou .ipa, executar um conjunto de testes padrão e, em seguida, visualizar os resultados.

> **Note**
>
> O Device Farm está disponível somente na região `us-west-2` (Oregon) da AWS .

Tópicos

- Pré-requisitos
- Etapa 1: Fazer login no console do
- Etapa 2: criar um projeto
- Etapa 3: Criar e iniciar uma execução
- Etapa 4: visualizar os resultados da execução
- Próximas etapas

Pré-requisitos

Antes de começar, verifique se você atendeu aos seguintes requisitos:

- Siga as etapas em Configuração. Você precisa de uma AWS conta e de um usuário AWS Identity and Access Management (IAM) com permissão para acessar o Device Farm.

- Para Android, você pode trazer um arquivo.apk (pacote de aplicativos Android) ou usar o aplicativo de amostra que fornecemos. Para iOS, você precisa de um arquivo .ipa (arquivo de aplicativo iOS). Você carregará o arquivo no Device Farm mais adiante neste passo a passo.

  > **Note**
  >
  > Confirme se o arquivo .ipa foi desenvolvido para um dispositivo iOS e não para um simulador.

- (Opcional) Você precisa de um teste de uma das estruturas de teste compatíveis com o Device Farm. Você carrega esse pacote de teste no Device Farm e, em seguida, executa o teste mais adiante neste passo a passo. Se você não tiver um pacote de testes disponível, poderá especificar e executar um conjunto de testes integrado padrão. Para obter mais informações, consulte Estruturas de teste e testes integrados no AWS Device Farm.

Etapa 1: Fazer login no console do

Você pode usar o console do Device Farm para criar e gerenciar projetos e execuções para testes. Você conhecerá projetos e execuções ainda nesta demonstração.

- Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.

Etapa 2: criar um projeto

Para testar um aplicativo no Device Farm, você deve primeiro criar um projeto.

1. No painel de navegação, escolha Teste para dispositivos móveis e Projetos.
2. Em Projetos de teste de dispositivos móveis, escolha Criar projeto.
3. Em Criar projeto, insira um Nome do projeto (por exemplo, **`MyDemoProject`**).
4. Escolha Criar.

   O console abre a página Testes automatizados do seu projeto recém-criado.

Etapa 3: Criar e iniciar uma execução

Agora que você já tem um projeto, pode criar e iniciar uma execução. Para obter mais informações, consulte Execuções.

1. Na guia Testes automatizados, escolha Criar execução. Como alternativa, você pode seguir o tutorial no console selecionando Criar execução com tutorial.
2. (Opcional) Em Configurações de execução, na seção Nome da execução, insira um nome para sua execução. Se nenhum nome for fornecido, o console do Device Farm nomeará sua execução como “My Device Farm run” por padrão.
3. Em Configurações de execução, na seção Tipo de execução, selecione seu tipo de execução. Selecione aplicativo Android se você não tiver um aplicativo pronto para teste ou se estiver

testando um aplicativo Android (.apk). Selecione o aplicativo iOS se você estiver testando um aplicativo iOS (.ipa).

4. Em Selecionar aplicativo, na seção Opções de seleção de aplicativos, escolha Selecionar aplicativo de amostra fornecido pelo Device Farm se você não tiver um aplicativo disponível para teste. Se você estiver trazendo seu próprio aplicativo, selecione Carregar próprio aplicativo e escolha o arquivo do seu aplicativo. Se você estiver fazendo upload de um aplicativo iOS, certifique-se de escolher iOS device (Dispositivo iOS), e não um simulador.
5. Em Configurar teste, na seção Selecionar estrutura de teste, escolha uma das estruturas de teste ou suítes de teste integradas. Para ter mais informações sobre cada opção, consulte Estruturas de teste e testes integrados.
   - Se você ainda não tiver empacotado seus testes para o Device Farm, escolha Integrado: fuzzing para executar um conjunto de testes padrão integrado. Você pode manter os valores padrão para Contagem de eventos, Restrição de eventos e Semente aleatória. Para obter mais informações, consulte the section called “Integrado: Fuzz (Android e iOS)”.
   - Se você tiver um pacote de teste de uma das estruturas de teste compatíveis, escolha a estrutura de teste correspondente e, em seguida, faça o upload do arquivo que contém seus testes.
6. Em Selecionar dispositivos, escolha Usar pool de dispositivos e Principais dispositivos.
7. (Opcional) Para adicionar configurações adicionais, abra o menu suspenso Configuração adicional. Nesta seção, você pode fazer qualquer um dos seguintes:

   - Para fornecer outros dados para o Device Farm usar durante a execução, ao lado de Adicionar dados extras, selecione Escolher arquivo e, em seguida, navegue até o arquivo .zip que contém os dados e escolha-o.
   - Para instalar uma aplicação adicional para o Device Farm usar durante a execução, ao lado de Instalar outras aplicações, selecione Escolher arquivo e, em seguida, procure e escolha o arquivo .apk ou .ipa que contém a aplicação. Repita isso para outros aplicativos que você deseja instalar. Você pode alterar a ordem de instalação arrastando e soltando os aplicativos depois de fazer upload deles.
   - Para especificar se Wi-Fi, Bluetooth, GPS ou NFC estará habilitado durante a execução, ao lado de Set radio states (Definir estados de rádio), selecione as caixas apropriadas.
   - Para predefinir a latitude e a longitude do dispositivo para a execução, ao lado de Device location (Local do dispositivo), insira as coordenadas.
   - Para predefinir a localidade da execução, em Localidade do dispositivo, escolha a localidade.
   - Selecione Habilitar a gravação de vídeo para gravar vídeos durante o teste.

- Selecione Habilitar a captura de dados de performance da aplicação para capturar dados de desempenho do dispositivo.

> **Note**
>
> Definir o estado do rádio e a localidade do dispositivo são opções disponíveis apenas para testes nativos do Android no momento.

> **Note**
>
> Se você tiver dispositivos privados, a configuração específica para dispositivos privados também será exibida.

8. Na parte inferior da página, escolha Criar execução para agendar a execução.

O Device Farm inicia a execução assim que os dispositivos estão disponíveis, normalmente em poucos minutos. Para ver o status da execução, na página Testes automatizados do seu projeto, escolha o nome da execução. Na página de execução, em Dispositivos, cada dispositivo começa com o ícone pendente

na tabela de dispositivos e depois muda para o ícone em execução

quando o teste começa. Quando cada teste termina, o console exibe um ícone de resultado do teste ao lado do nome do dispositivo. Quando todos os testes são concluídos, o ícone pendente ao lado da execução muda para um ícone de resultado de teste.

Etapa 4: visualizar os resultados da execução

Para ver os resultados do teste, na página Testes automatizados do seu projeto, escolha o nome da execução. A página de resumo exibe:

- O número total de testes, por resultado.
- Lista de testes com avisos exclusivos ou falhas.
- Uma lista de dispositivos com resultados de testes para cada um.

- Quaisquer capturas de tela durante a execução, agrupadas por dispositivo.
- Uma seção para baixar o resultado da análise.

Para obter mais informações, consulte Visualizando relatórios de teste no Device Farm.

Próximas etapas

Para obter mais informações sobre o Device Farm, consulte Conceitos.

Comprar um slot de dispositivo no Device Farm

Você pode usar o console Device Farm, AWS Command Line Interface (AWS CLI) ou a API Device Farm para comprar um slot de dispositivo.

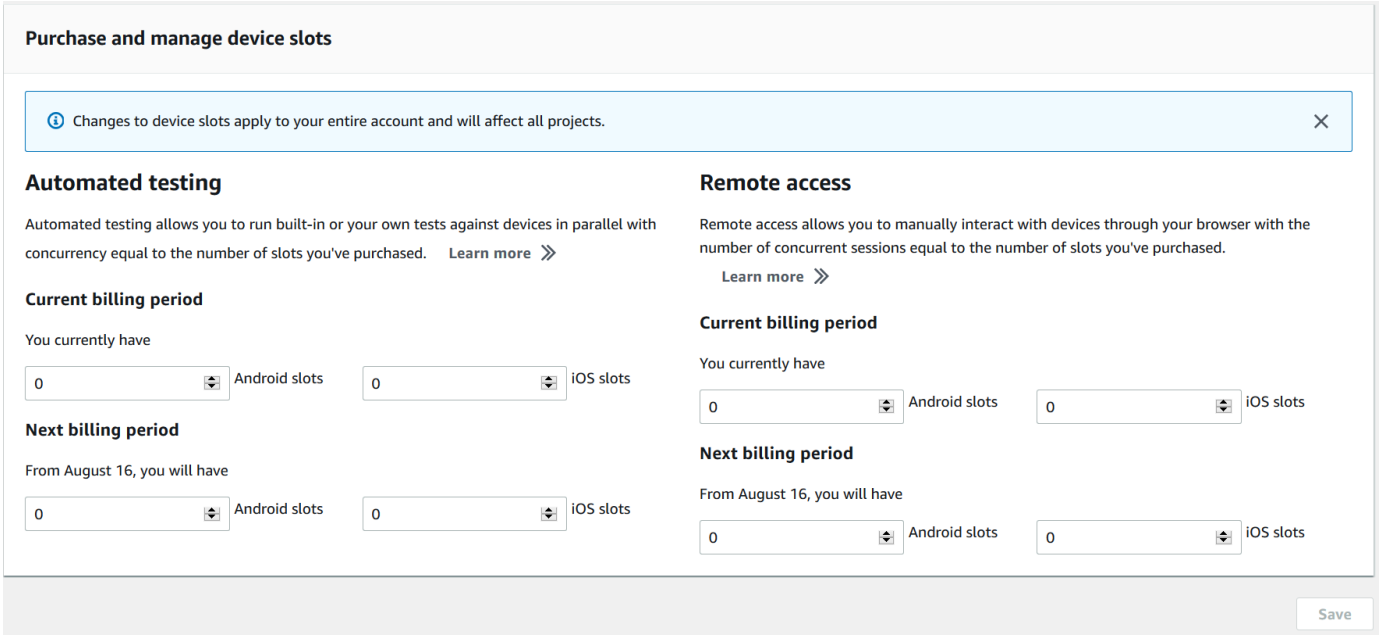
Comprar slots de dispositivos (console)

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. No painel de navegação, escolha Teste para dispositivos móveis e Slots de dispositivos.
3. Na página Adquira e gerencie slots de dispositivos, você pode criar seu próprio pacote personalizado escolhendo o número de slots de dispositivos de Teste automatizado e Acesso remoto que deseja comprar. Especifique os valores dos slots para o período de cobrança atual e o próximo.

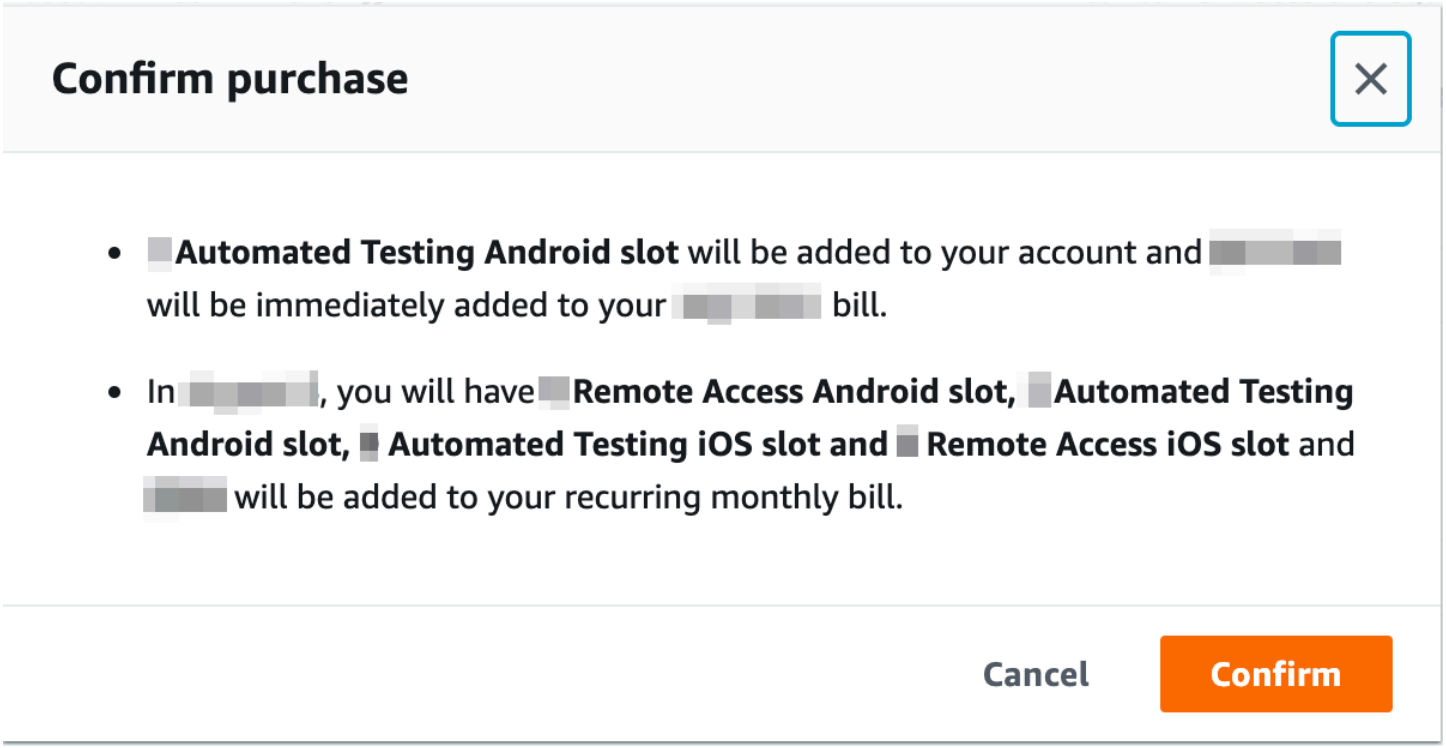
   Conforme você altera o valor do slot, o texto é atualizado dinamicamente com o valor do faturamento. Para obter mais informações, consulte os preços do AWS Device Farm.

   > **Important**
   >
   > Se você alterar o número de slots de dispositivos, mas receber uma mensagem de contato ou de contato para comprar, sua AWS conta ainda não foi aprovada para comprar o número de slots de dispositivo que você solicitou.
   > Essas opções solicitam que você envie um e-mail para a equipe de suporte do Device Farm. No e-mail, especifique o número de cada tipo de dispositivo que você deseja comprar e para qual ciclo de cobrança.

   > **Note**
   >
   > As alterações nos slots do dispositivo se aplicam a toda a sua conta e afetam todos os projetos.



4. Escolha Purchase (Comprar). A janela Confirmar compra será exibida. Revise as informações e escolha Confirmar para concluir a transação.



Na página Adquira e gerencie slots de dispositivos, você pode ver o número de slots de dispositivos que você tem atualmente. Se tiver aumentado ou diminuído o número de slots, você verá o número de slots que terá um mês depois da data em que fez a alteração.

Comprar um slot de dispositivo (AWS CLI)

Você pode executar o comando purchase-offering para comprar uma oferta.

Para listar as configurações de sua conta do Device Farm, incluindo o número máximo de slots de dispositivo que você pode comprar e o número de minutos de avaliação gratuita restantes, execute o comando get-account-settings. Você verá algo semelhante ao resultado a seguir:

```
{
    "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
        }
    }
}
```

Para listar as ofertas de slot para dispositivos disponíveis para você, execute o comando list-offerings. Você deve ver uma saída semelhante a:

```
{
```

```
    "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"
        }
    ]
}
```

Para listar as promoções de ofertas disponíveis, execute o comando list-offering-promotions.

> **Note**
>
> Esse comando retorna apenas as promoções que você ainda não comprou. Assim que você comprar um ou mais slots em qualquer oferta usando uma promoção, essa promoção deixará de ser exibida nos resultados.

Você deve ver uma saída semelhante a:

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

Para obter o status da oferta, execute o comando get-offering-status. Você deve ver uma saída semelhante a:

```
{
    "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
        }
    }
```

```
}
```

Os comandos renew-offering e list-offering-transactions também estão disponíveis para esse recurso. Para obter mais informações, consulte AWS CLI referência.

Comprar um slot de dispositivo (API)

1. Ligue para a GetAccountSettingsoperação para listar as configurações da sua conta.
2. Ligue para a ListOfferingsoperação para listar as ofertas de slots de dispositivos disponíveis para você.
3. Ligue para a ListOfferingPromotionsoperação para listar as promoções de oferta que estão disponíveis.

   > **Note**
   >
   > Esse comando retorna apenas as promoções que você ainda não comprou. Assim que você comprar um ou mais slots usando uma promoção de oferta, essa promoção deixará de ser exibida nos resultados.

4. Ligue para a PurchaseOfferingoperação para comprar uma oferta.
5. Ligue para a GetOfferingStatusoperação para obter o status da oferta.

Os comandos RenewOffering e ListOfferingTransactions também estão disponíveis para esse recurso.

Para obter informações sobre como usar a API do Device Farm, consulte Automatização do Device Farm.

Cancelamento de um slot de dispositivo no Device Farm

Você pode cancelar o número de slots de dispositivos para testes automatizados e acesso remoto. Para obter instruções, consulte uma das seções a seguir. O valor cobrado em sua conta para o próximo ciclo de cobrança será listado abaixo do campo do período de cobrança.

Para obter mais informações sobre slots de dispositivos, consulteComprar um slot de dispositivo no Device Farm.

Cancelar um slot de dispositivo (console)

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. No painel de navegação, escolha Teste para dispositivos móveis e Slots de dispositivos.
3. Na página Comprar e gerenciar slots de dispositivos, você pode diminuir o número de slots de dispositivos para testes automatizados e acesso remoto diminuindo o valor em Próximo período de cobrança. O valor cobrado em sua conta para o próximo ciclo de cobrança será listado abaixo do campo do período de cobrança.
4. Escolha Salvar. Uma janela Confirmar alteração será exibida. Revise as informações e escolha Confirmar para concluir a transação.

Cancelar um slot de dispositivo (AWS CLI)

Você pode executar o renew-offering comando para alterar a quantidade de dispositivos para o próximo ciclo de cobrança.

Cancelar um slot de dispositivo (API)

Ligue para a RenewOfferingoperação para alterar a quantidade de dispositivos em sua conta.

Conceitos do AWS Device Farm

O Device Farm é um serviço de teste de aplicações que você pode usar para testar e interagir com suas aplicações Android, iOS e web em telefones e tablets reais e físicos hospedados pela Amazon Web Services (AWS).

Esta seção descreve conceitos importantes do Device Farm.

- Suporte de dispositivos no AWS Device Farm
- Ambientes de teste no AWS Device Farm
- Execuções
- Apps
- Relatórios no AWS Device Farm
- Sessões

Para obter mais informações sobre os tipos de teste compatíveis no Device Farm, consulte Estruturas de teste e testes integrados no AWS Device Farm.

Suporte de dispositivos no AWS Device Farm

As seções a seguir fornecem informações sobre o suporte a dispositivos no Device Farm.

Tópicos

- Dispositivos compatíveis
- Grupos de dispositivos
- Dispositivos privados
- Marcas de dispositivo
- Slots para dispositivo
- Aplicações de dispositivos pré-instalados
- Recursos dos dispositivos

Dispositivos compatíveis

O Device Farm oferece suporte a centenas de combinações de sistemas operacionais e dispositivos Android e iOS exclusivos e populares. A lista de dispositivos disponíveis cresce à medida que novos dispositivos entram no mercado. Para a lista completa de dispositivos, consulte Lista de dispositivos.

Grupos de dispositivos

O Device Farm organiza seus dispositivos em pools de dispositivos que você pode usar para seus testes. Esses pools de dispositivos contêm dispositivos relacionados, como dispositivos que são executados somente no Android ou somente no iOS. O Device Farm fornece pools de dispositivos selecionados, como os dos principais dispositivos. Você também pode criar grupos de dispositivos que combinam dispositivos públicos e privados.

Dispositivos privados

Os dispositivos privados permitem especificar configurações de hardware e software exatas para as suas necessidades de testes. Certas configurações, como dispositivos Android com root, podem ser permitidas como dispositivos privados. Cada dispositivo privado é um dispositivo físico que o Device Farm implementa em seu nome em um data center da Amazon. Os seus dispositivos privados estão disponíveis exclusivamente para os testes automatizados e manuais. Depois que você optar por encerrar a sua assinatura, o hardware será removido do ambiente. Para obter mais informações, consulte Dispositivos privados e Dispositivos privados no AWS Device Farm.

Marcas de dispositivo

O Device Farm executa testes em dispositivos móveis e tablets físicos a partir de uma variedade de OEMs.

Slots para dispositivo

Os slots para dispositivo correspondem à simultaneidade com que o número de slots para dispositivo que você adquiriu determina o número de dispositivos que você pode executar em testes ou sessões de acesso remoto.

Existem dois tipos de slots de dispositivos:

- O slot de dispositivo de acesso remoto é aquele em que você pode executar sessões de acesso remoto simultaneamente.

Se você tiver um único slot para dispositivo de acesso remoto, poderá executar somente uma sessão de acesso remoto por vez. Se comprar mais slots de dispositivos de teste remoto, você poderá executar várias sessões simultaneamente.

- O slot de dispositivo de teste automatizado é aquele em que você pode executar testes simultaneamente.

  Se tiver um slot de dispositivo de teste automatizado, você só poderá executar testes em um dispositivo por vez. Se comprar mais slots de dispositivos de teste automatizado, você poderá executar vários testes simultaneamente, em vários dispositivos, para obter os resultados mais rapidamente.

Você pode comprar slots para dispositivo com base na família do dispositivo (dispositivos Android ou iOS para testes automatizados e dispositivos Android ou iOS para acesso remoto). Para obter mais informações, consulte Definição de preço do Device Farm.

Aplicações de dispositivos pré-instalados

Os dispositivos no Device Farm incluem um pequeno número de aplicações que já estão instaladas pelos fabricantes e operadoras.

Recursos dos dispositivos

Todos os dispositivos têm conexão Wi-Fi com a Internet. Ele não têm conexão com as operadores e não podem fazer ligações telefônicas nem enviar mensagens SMS.

Você pode tirar fotos com qualquer dispositivo que tenha câmera frontal ou traseira. Por causa da maneira como os dispositivos são montados, as fotos podem ter uma aparência escura e tremida.

O Google Play Services está instalado nos dispositivos com câmera, mas esses dispositivos não têm uma conta Google ativa.

Ambientes de teste no AWS Device Farm

A AWS Device Farm fornece ambientes de teste padrão e personalizados para a execução de testes automatizados. Você pode escolher um ambiente de teste personalizado para controle total sobre os testes automatizados. Ou você pode escolher o ambiente de teste padrão do Device Farm, que oferece relatórios detalhados de cada teste em seu conjunto de testes automatizados.

Tópicos

- Ambiente de teste padrão
- Ambiente de teste personalizado

Ambiente de teste padrão

Quando você executa um teste no ambiente padrão, o Device Farm fornece logs e relatórios detalhados para cada caso no seu conjunto de testes. Você pode visualizar dados de desempenho, vídeos, capturas de tela e logs para cada teste a fim de identificar e corrigir problemas no aplicativo.

> **Note**
>
> Como o Device Farm fornece relatórios granulares no ambiente padrão, os tempos de execução dos testes podem ser mais longos do que quando você os executa localmente. Se você quiser tempos de execução menores, execute os testes em um ambiente de teste personalizado.

Ambiente de teste personalizado

Ao personalizar o ambiente de teste, você pode especificar os comandos que o Device Farm deve executar para realizar seus testes. Isso garante que os testes no Device Farm sejam executados de forma semelhante aos testes executados em seu computador local. Executar os testes nesse modo também permite que o streaming de vídeo ao vivo e log dos testes. Ao executar testes em um ambiente de teste personalizado, você não recebe relatórios granulares para cada caso de teste. Para obter mais informações, consulte Ambientes de teste personalizados no AWS Device Farm.

Você tem a opção de usar um ambiente de teste personalizado ao usar o console do Device Farm, a AWS CLI ou a API do Device Farm para criar uma execução de teste.

Para obter mais informações, consulte Uploading a Custom Test Spec Using the AWS CLI e Criando uma execução de teste no Device Farm.

Executa no AWS Device Farm

As seções a seguir contêm informações sobre execuções no Device Farm.

Uma execução no Device Farm representa uma compilação específica da aplicação, com um conjunto específico de testes, a ser executada em um conjunto específico de dispositivos. Uma execução produz um relatório que contém informações sobre os resultados da execução. A execução contém um ou mais trabalhos.

Tópicos

- Configuração da execução
- Retenção de arquivos de execução
- Estado do dispositivo de execução
- Execuções paralelas
- Configurar o tempo limite de execução
- Anúncios nas execuções
- Mídias nas execuções
- Tarefas comuns nas execuções

Configuração da execução

Como parte de uma execução, você pode fornecer configurações que o Device Farm pode usar para substituir as configurações atuais do dispositivo. Isso inclui coordenadas de latitude e longitude, local, especificações de rádio (como Bluetooth, GPS, NFC e Wi-Fi), dados extras (contidos em um arquivo .zip) e aplicativos auxiliares (aplicativos que devem ser instalados antes do aplicativo que será testado).

Retenção de arquivos de execução

O Device Farm armazena aplicações e arquivos por 30 dias e depois os exclui do sistema. No entanto, você mesmo pode excluir seus arquivos a qualquer momento.

O Device Farm armazena seus resultados de execução, logs e capturas de tela por 400 dias e depois os exclui do sistema.

Estado do dispositivo de execução

O Device Farm sempre reinicia um dispositivo antes de disponibilizá-lo para o próximo trabalho.

Execuções paralelas

O Device Farm executa testes em paralelo à medida que os dispositivos ficam disponíveis.

Configurar o tempo limite de execução

Você pode definir por quanto tempo um teste deve ser executado antes de interromper a execução de teste de cada dispositivo. Por exemplo, se a conclusão dos testes demorar 20 minutos por dispositivo, você deve escolher um tempo limite de 30 minutos por dispositivo.

Para obter mais informações, consulte Definindo o tempo limite de execução para execuções de teste no AWS Device Farm.

Anúncios nas execuções

Recomendamos que você remova os anúncios de suas aplicações antes de carregá-las no Device Farm. Não podemos garantir que os anúncios sejam exibidos durante execuções.

Mídias nas execuções

Você pode fornecer mídias ou outros dados para acompanhar seu aplicativo. Os dados adicionais devem ser fornecidos em um arquivo .zip com tamanho não superior a 4 GB.

Tarefas comuns nas execuções

Para ter mais informações, consulte Criando uma execução de teste no Device Farm e Testes são executados no AWS Device Farm.

Aplicativos no AWS Device Farm

As seções a seguir contêm informações sobre o comportamento dos aplicativos no Device Farm.

Tópicos

- Instrumentação de aplicações
- Nova assinatura de aplicações nas execuções
- Aplicações ofuscadas nas execuções

Instrumentação de aplicações

Não é necessário instrumentar suas aplicações ou fornecer ao Device Farm o código-fonte delas. As aplicações Android podem ser enviadas não modificadas. As aplicações iOS devem ser compilados com o destino Dispositivo iOS, em vez do simulador.

Nova assinatura de aplicações nas execuções

Para aplicativos iOS, você não precisa adicionar nenhum Device Farm UUIDs ao seu perfil de aprovisionamento. O Device Farm substitui o perfil de provisionamento incorporado por um perfil curinga e, em seguida, assina novamente a aplicação. Se você fornecer dados auxiliares, o Device Farm os adicionará ao pacote do aplicativo antes de instalá-lo, para que o auxiliar exista no sandbox da aplicação. A reassinatura do aplicativo remove direitos como Grupo de aplicativos, domínios associados, Game Center,,, Configuração de acessórios sem fio HealthKit HomeKit, compra no aplicativo, áudio entre aplicativos, Apple Pay, notificações push e configuração e controle de VPN.

Para aplicações Android, o Device Farm assina novamente a aplicação. Isso pode interromper qualquer funcionalidade que dependa da assinatura do aplicativo, como a API do Google Maps para Android, ou pode acionar a detecção antipirataria ou antiadulteração em produtos como. DexGuard

Aplicações ofuscadas nas execuções

Para aplicativos Android, se o aplicativo estiver ofuscado, você ainda poderá testá-lo com o Device Farm se você usar. ProGuard No entanto, se você usar DexGuard com medidas antipirataria, o Device Farm não poderá assinar novamente e executar testes no aplicativo.

Relatórios no AWS Device Farm

As seções a seguir fornecem informações sobre os relatórios de teste do Device Farm.

Tópicos

- Retenção de relatório
- Componentes do relatório
- Logs nos relatórios
- Tarefas comuns relacionadas aos relatórios

Retenção de relatório

O Device Farm armazena seus relatórios por 400 dias. Esses relatórios incluem metadados, logs, capturas de tela e dados de desempenho.

Componentes do relatório

Os relatórios no Device Farm contêm informações de aprovação e reprovação, relatórios de falhas, logs de testes e dispositivos, capturas de tela e dados de desempenho.

Os relatórios incluem dados detalhados por dispositivo e resultados técnicos, como o número de ocorrências de um determinado problema.

Logs nos relatórios

Os relatórios incluem capturas de logcat para testes do Android e logs completos do console de dispositivo para testes de iOS.

Tarefas comuns relacionadas aos relatórios

Para obter mais informações, consulte Visualizando relatórios de teste no Device Farm.

Sessões no AWS Device Farm

Você pode usar o Device Farm para realizar testes interativos de aplicações para Android e iOS por meio de sessões de acesso remoto em um navegador da web. Esse tipo de teste interativo ajuda a dar suporte a engenheiros em uma chamada do cliente para percorrer, passo a passo, o problema do cliente. Os desenvolvedores podem reproduzir um problema em um dispositivo específico para isolar as possíveis causas do problema. Você pode usar sessões remotas para realizar testes de usabilidade com seus clientes-alvo.

Tópicos

- Dispositivos compatíveis com acesso remoto
- Retenção de arquivos de sessão
- Instrumentação de aplicações
- Nova assinatura de aplicações nas sessões
- Aplicações ofuscadas nas sessões

Dispositivos compatíveis com acesso remoto

O Device Farm é compatível com uma série de dispositivos Android e iOS exclusivos e populares. A lista de dispositivos disponíveis cresce à medida que novos dispositivos entram no mercado. O console do Device Farm exibe a lista atual de dispositivos Android e iOS disponíveis para acesso remoto. Para obter mais informações, consulte Suporte de dispositivos no AWS Device Farm.

Retenção de arquivos de sessão

O Device Farm armazena aplicações e arquivos por 30 dias e, depois, os exclui do sistema. No entanto, você mesmo pode excluir seus arquivos a qualquer momento.

O Device Farm armazena logs de sessão e vídeos capturados por 400 dias e, depois, os exclui do sistema.

Instrumentação de aplicações

Não é necessário instrumentar suas aplicações ou fornecer ao Device Farm o código-fonte delas. Os aplicativos Android e iOS podem ser enviados sem alteração.

Nova assinatura de aplicações nas sessões

O Device Farm assina novamente as aplicações para Android e iOS. Ele pode interromper a funcionalidade dependente da assinatura do aplicativo. Por exemplo, a API do Google Maps para Android depende da assinatura do aplicativo. A reassinatura do aplicativo também pode acionar a detecção antipirataria ou antiadulteração em produtos como dispositivos Android. DexGuard

Aplicações ofuscadas nas sessões

Para aplicativos Android, se o aplicativo estiver ofuscado, você ainda poderá testá-lo com o Device Farm se você usar. ProGuard No entanto, se você usar DexGuard com medidas antipirataria, o Device Farm não poderá assinar novamente o aplicativo.

Projetos no AWS Device Farm

Um projeto no Device Farm representa um espaço de trabalho lógico no Device Farm que contém execuções, uma execução para cada teste de um único aplicativo em um ou mais dispositivos. Os projetos permitem que você organize os espaços de trabalho da maneira que preferir. Por exemplo, pode haver um projeto por título de aplicativo ou pode haver um projeto por plataforma. Você pode criar quantos projetos necessitar.

Você pode usar o console do AWS Device Farm, AWS Command Line Interface (AWS CLI) ou a API do AWS Device Farm para trabalhar com projetos.

Tópicos

- Criação de um projeto no AWS Device Farm
- Visualização da lista de projetos no AWS Device Farm

Criação de um projeto no AWS Device Farm

Você pode criar um projeto usando o console do AWS Device Farm ou a API do AWS Device Farm. AWS CLI

Pré-requisitos

- Siga as etapas em Configuração.

Criar um projeto (console)

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e, em seguida, Projetos.
3. Selecione New project (Novo projeto).
4. Insira um nome para seu projeto e escolha Enviar.
5. Para especificar configurações para o projeto, escolha Project settings (Configurações do projeto). Essas configurações incluem o tempo limite padrão para execuções de teste. Depois que as configurações forem aplicadas, elas serão usadas em todas as execuções de teste

do projeto. Para obter mais informações, consulte Definindo o tempo limite de execução para execuções de teste no AWS Device Farm.

Criar um projeto (AWS CLI)

- Execute create-project especificando o nome do projeto.

  Exemplo: .

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

  A AWS CLI resposta inclui o Amazon Resource Name (ARN) do projeto.

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

  Para ter mais informações, consulte create-project e AWS CLI referência.

Criar um projeto (API)

- Chame a API `CreateProject`.

Para obter informações sobre como usar a API do Device Farm, consulte Automatização do Device Farm.

Visualização da lista de projetos no AWS Device Farm

Você pode usar o console do AWS Device Farm ou a API do AWS Device Farm para visualizar a lista de projetos. AWS CLI

Tópicos

- Pré-requisitos
- Visualizar a lista de projetos (console)
- Visualizar a lista de projetos (AWS CLI)
- Visualizar a lista de projetos (API)

Pré-requisitos

- Crie pelo menos um projeto no Device Farm. Siga as instruções em Criação de um projeto no AWS Device Farm e retorne para esta página.

Visualizar a lista de projetos (console)

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. Para localizar a lista de projetos disponíveis, faça o seguinte:
   - Para projetos de teste de dispositivos móveis, no menu de navegação do Device Farm, escolha Teste para dispositivos móveis e escolha Projetos.
   - Para projetos de teste de navegadores de desktop, no menu de navegação do Device Farm, escolha Teste para navegadores de área de trabalho e Projetos.

Visualizar a lista de projetos (AWS CLI)

- Para visualizar a lista de projetos, execute o comando list-projects.

  Para visualizar informações sobre um único projeto, execute o comando get-project.

Para obter informações sobre como usar o Device Farm com o AWS CLI, consulteAWS CLI referência.

Visualizar a lista de projetos (API)

- Para visualizar a lista de projetos, execute a API `ListProjects`.

  Para visualizar informações sobre um único projeto, chame a API `GetProject`.

Para obter informações sobre a API do AWS Device Farm, consulte Automatização do Device Farm.

Testes são executados no AWS Device Farm

Uma execução no Device Farm representa uma compilação específica da aplicação, com um conjunto específico de testes, a ser executada em um conjunto específico de dispositivos. Uma execução produz um relatório que contém informações sobre os resultados da execução. A execução contém um ou mais trabalhos. Para obter mais informações, consulte Execuções.

Você pode usar o console do AWS Device Farm, AWS Command Line Interface (AWS CLI) ou a API do AWS Device Farm para trabalhar com execuções de teste.

Tópicos

- Criando uma execução de teste no Device Farm
- Definindo o tempo limite de execução para execuções de teste no AWS Device Farm
- Simulação de conexões e condições de rede para suas execuções do AWS Device Farm
- Interrompendo uma execução no AWS Device Farm
- Visualizando uma lista de execuções no AWS Device Farm
- Criação de um pool de dispositivos no AWS Device Farm
- Análise de resultados de testes no AWS Device Farm

Criando uma execução de teste no Device Farm

Você pode usar o console Device Farm ou AWS CLI a API Device Farm para criar uma execução de teste. Você também pode usar um plug-in compatível, como os plug-ins Jenkins ou Gradle para o Device Farm. Para obter mais informações sobre plug-ins, consulte Ferramentas e plug-ins. Para obter informações sobre execuções, consulte Execuções.

Tópicos

- Pré-requisitos
- Criar uma execução de teste (console)
- Criar uma execução de teste (AWS CLI)
- Criar uma execução de teste (API)
- Próximas etapas

Pré-requisitos

Você deve ter um projeto no Device Farm. Siga as instruções em Criação de um projeto no AWS Device Farm e retorne para esta página.

Criar uma execução de teste (console)

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. No painel de navegação, escolha Teste para dispositivos móveis e Projetos.
3. Se já tiver um projeto, você poderá fazer upload dos testes para ele. Caso contrário, selecione Novo projeto, insira um Nome do projeto e escolha Criar.
4. Abra seu projeto e escolha Criar execução.
5. (Opcional) Em Configurações de execução, na seção Nome da execução, insira um nome para sua execução. Se nenhum nome for fornecido, o console do Device Farm nomeará sua execução como “My Device Farm run” por padrão.
6. (Opcional) Em Configurações de execução, na seção Tempo limite do trabalho, você pode especificar o tempo limite de execução para sua execução de teste. Se você estiver usando vagas de teste ilimitadas, confirme se Unmetered está selecionada em Método de cobrança.
7. Em Configurações de execução, na seção Tipo de execução, selecione seu tipo de execução. Selecione aplicativo Android se você não tiver um aplicativo pronto para teste ou se estiver testando um aplicativo Android (.apk). Selecione o aplicativo iOS se estiver testando um aplicativo iOS (.ipa). Selecione Aplicativo Web se quiser testar aplicativos Web.
8. Em Selecionar aplicativo, na seção Opções de seleção de aplicativos, escolha Selecionar aplicativo de amostra fornecido pelo Device Farm se você não tiver um aplicativo disponível para teste. Se você estiver trazendo seu próprio aplicativo, selecione Carregar próprio aplicativo e escolha o arquivo do seu aplicativo. Se você estiver fazendo upload de um aplicativo iOS, certifique-se de escolher iOS device (Dispositivo iOS), e não um simulador.
9. Em Configurar teste, escolha uma das estruturas de teste disponíveis.

> **Note**
>
> Se não tiver testes disponíveis, escolha Integrado: fuzzing para executar um pacote integrado padrão de testes. Se escolher Integrado: fuzzing e as caixas Contagem de eventos, Limite de eventos e Propagação aleatória forem exibidas, você poderá alterar ou manter os valores.

Para obter mais informações sobre pacotes de testes, consulte Estruturas de teste e testes integrados no AWS Device Farm.

10. Se você não escolheu Integrado: Fuzz, selecione Escolher arquivo em Selecionar pacote de teste. Procure e escolha o arquivo que contém seus testes.
11. Para seu ambiente de teste, escolha Executar teste em nosso ambiente padrão ou Executar teste em um ambiente personalizado. Para obter mais informações, consulte Ambientes de teste no AWS Device Farm.
12. Se você estiver usando um ambiente de teste personalizado, você pode, opcionalmente, fazer o seguinte:

    a. Se você quiser editar a especificação de teste padrão em um ambiente de teste personalizado, escolha Editar para atualizar a especificação YAML padrão.

    b. Se você fizer alterações na especificação de teste, escolha Salvar como novo para atualizar.

13. Em Selecionar dispositivos, faça o seguinte:

    - Para escolher um pool de dispositivos incorporado para executar os testes, em Grupo de dispositivos, escolha Principais dispositivos.
    - Para criar o próprio grupo de dispositivos para executar os testes, siga as instruções em Criação de um pool de dispositivos e retorne a esta página.
    - Se você tiver criado o próprio grupo de dispositivos anteriormente, em Grupo de dispositivos, escolha o grupo de dispositivos.
    - Selecione Selecionar dispositivos manualmente e escolha os dispositivos desejados com os quais você deseja correr. Essa configuração não será salva.

    Para obter mais informações, consulte Suporte de dispositivos no AWS Device Farm.

14. (Opcional) Para adicionar configurações adicionais, abra o menu suspenso Configuração adicional. Nesta seção, você pode fazer o seguinte:

    - Para fornecer outros dados para o Device Farm usar durante a execução, ao lado de Adicionar dados extras, selecione Escolher arquivo e, em seguida, navegue até o arquivo .zip que contém os dados e escolha-o.
    - Para instalar uma aplicação adicional para o Device Farm usar durante a execução, ao lado de Instalar outras aplicações, selecione Escolher arquivo e, em seguida, procure e escolha

o arquivo .apk ou .ipa que contém a aplicação. Repita isso para outros aplicativos que você deseja instalar. Você pode alterar a ordem de instalação arrastando e soltando os aplicativos depois de fazer upload deles.

- Para especificar se Wi-Fi, Bluetooth, GPS ou NFC estará habilitado durante a execução, ao lado de Set radio states (Definir estados de rádio), selecione as caixas apropriadas.
- Para predefinir a latitude e a longitude do dispositivo para a execução, ao lado de Device location (Local do dispositivo), insira as coordenadas.
- Para predefinir a localidade da execução, em Localidade do dispositivo, escolha a localidade.
- Selecione Habilitar a gravação de vídeo para gravar vídeos durante o teste.
- Selecione Habilitar a captura de dados de performance da aplicação para capturar dados de desempenho do dispositivo.

> **Note**
>
> Definir o estado do rádio e a localidade do dispositivo são opções disponíveis apenas para testes nativos do Android no momento.

> **Note**
>
> Se você tiver dispositivos privados, a configuração específica para dispositivos privados também será exibida.

15. Na parte inferior da página, escolha Criar execução para agendar a execução.

O Device Farm inicia a execução assim que os dispositivos estão disponíveis, normalmente em poucos minutos. Durante a execução do teste, o console do Device Farm exibe um ícone pendente

na tabela de execução. Cada dispositivo na execução também começará com o ícone pendente e, em seguida, mudará para o ícone em execução

quando o teste começar. Quando cada teste é concluído, um ícone de resultado do teste é exibido ao lado do nome do dispositivo. Quando todos os testes tiverem sido concluídos, o ícone pendente ao lado da execução mudará para um ícone de resultado de teste.

Se você quiser interromper a execução do teste, consulte Interrompendo uma execução no AWS Device Farm.

Criar uma execução de teste (AWS CLI)

Você pode usar o AWS CLI para criar uma execução de teste.

Tópicos

- Etapa 1: escolher um projeto
- Etapa 2: escolher um grupo de dispositivos
- Etapa 3: fazer upload do arquivo da aplicação
- Etapa 4: fazer upload do pacote de scripts de teste
- Etapa 5: (opcional) fazer upload de sua especificação de teste personalizada
- Etapa 6: programar uma execução de teste

Etapa 1: escolher um projeto

Você deve associar sua execução de teste a um projeto do Device Farm.

1. Para listar seus projetos do Device Farm, execute list-projects. Se você não tiver um projeto, consulte Criação de um projeto no AWS Device Farm.

   Exemplo:

   ```
   aws devicefarm list-projects
   ```

   A resposta inclui uma lista de seus projetos do Device Farm.

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

2. Escolha um projeto a ser associado à execução de teste e anote seu nome de recurso da Amazon (ARN).

Etapa 2: escolher um grupo de dispositivos

Você deve escolher um grupo de dispositivos a ser associado à execução de teste.

1. Para visualizar os grupos de dispositivos, execute list-device-pools especificando o ARN do projeto.

   Exemplo:

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

   A resposta inclui os grupos de dispositivos integrados do Device Farm, como Top Devices, e os grupos de dispositivos criados anteriormente para esse projeto:

   ```
   {
       "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. Escolha um grupo de dispositivos e anote o ARN.

   Você também pode criar um grupo de dispositivos e retornar a essa etapa. Para obter mais informações, consulte Criar um grupo de dispositivos (AWS CLI).

Etapa 3: fazer upload do arquivo da aplicação

Para criar sua solicitação de upload e obter um URL de upload pré-assinado do Amazon Simple Storage Service (Amazon S3), você precisa:

- O ARN do projeto.
- O nome do arquivo do aplicativo.
- O tipo do upload.

Para obter mais informações, consulte create-upload.

1. Para fazer upload de um arquivo, execute create-upload com os parâmetros `--project-arn`, `--name` e `--type`.

   Este exemplo cria um upload para um aplicativo Android:

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

   A resposta inclui o ARN de upload do aplicativo e um URL pré-assinado.

   ```
   {
       "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. Anote o ARN de upload do aplicativo e o URL pré-assinado.
3. Faça o upload do arquivo da aplicação usando o URL predefinido do Amazon S3. Este exemplo usa curl para fazer upload de um arquivo .apk do Android:

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

   Para obter mais informações, consulte Carregar objetos usando presigned URLs no Guia do usuário do Amazon Simple Storage Service.
4. Para verificar o status de upload do aplicativo, execute get-upload e especifique o ARN de upload do aplicativo.

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

   Aguarde até o status na resposta ser SUCCEEDED para fazer upload do pacote de scripts de teste.

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

Etapa 4: fazer upload do pacote de scripts de teste

Em seguida, você faz upload do pacote de scripts de teste.

1. Para criar sua solicitação de upload e obter um URL de upload pré-assinado do Amazon S3, execute create-upload com os parâmetros `--project-arn`, `--name` e `--type`.

   Este exemplo cria um upload do pacote de testes do Appium Java TestNG:

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

   A resposta inclui o ARN de upload do pacote de testes e um URL pré-assinado.

   ```
   {
       "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. Anote o ARN de upload do pacote de testes e o URL pré-assinado.
3. Faça upload do arquivo do pacote de scripts de teste usando o URL pré-assinado do Amazon S3. Este exemplo usa curl para fazer upload de um arquivo de scripts do Appium TestNG compactado:

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

4. Para verificar o status do upload do pacote de scripts de teste, execute get-upload e especifique o ARN de upload do pacote de testes da etapa 1.

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

Aguarde o status na resposta ser SUCCEEDED para avançar à próxima etapa, opcional.

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

Etapa 5: (opcional) fazer upload de sua especificação de teste personalizada

Se você estiver executando os testes em um ambiente de teste padrão, ignore esta etapa.

O Device Farm mantém um arquivo de especificações de teste padrão para cada tipo de teste compatível. Em seguida, você faz download da especificação de teste padrão e o usa para criar o upload da especificação de teste personalizada para executar os testes em um ambiente de teste personalizado. Para obter mais informações, consulte Ambientes de teste no AWS Device Farm.

1. Para encontrar o ARN de upload para a especificação de teste padrão, execute list-uploads e especifique o ARN do projeto.

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

A resposta contém uma entrada para cada especificação de teste padrão:

```
{
    "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. Escolha a especificação de teste padrão na lista. Anote o ARN do upload.
3. Para fazer download da especificação de teste padrão, execute get-upload e especifique o ARN de upload.

   Exemplo:

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

   A resposta contém um URL pré-assinado em que você pode fazer download da especificação de teste padrão.
4. Este exemplo usa curl para fazer download da especificação de teste padrão e salvá-lo como `MyTestSpec.yml`:

   ```
   curl "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/ExampleURL" >
    MyTestSpec.yml
   ```
5. Você pode editar a especificação de teste padrão para atender aos requisitos de teste e, em seguida, usar a especificação de teste modificada em execuções de teste futuras. Ignore esta etapa para usar a especificação de teste padrão no estado em que ela se encontra em um ambiente de teste personalizado.
6. Para criar um upload da especificação de teste personalizada, execute create-upload especificando o nome e o tipo da especificação de teste e o ARN do projeto.

   Este exemplo cria um upload para uma especificação de teste personalizada do Appium Java TestNG:

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

A resposta inclui o ARN de upload da especificação de teste e um URL pré-assinado:

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

7. Anote o ARN do upload da especificação de teste e o URL pré-assinado.
8. Faça upload do arquivo de especificações de teste usando o URL pré-assinado do Amazon S3. Este exemplo é usado curl para carregar uma especificação de teste do Appium JavaTest NG:

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

9. Para verificar o status de upload da especificação de teste, execute get-upload e especifique o ARN de upload.

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

Aguarde até o status na resposta ser SUCCEEDED antes de programar a execução de teste.

```
{
    "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}"
```

```
    }
}
```

Para atualizar a especificação de teste personalizada, execute update-upload especificando o ARN de upload para a especificação de teste. Para obter mais informações, consulte update-upload.

Etapa 6: programar uma execução de teste

Para agendar uma execução de teste com o AWS CLI, executeschedule-run, especificando:

- O ARN do projeto da etapa 1.
- O ARN do grupo de dispositivos da etapa 2.
- O ARN de upload do aplicativo da etapa 3.
- O ARN de upload do pacote de testes da etapa 4.

Se estiver executando testes em um ambiente de teste personalizado, você também precisará do ARN da especificação de teste da etapa 5.

Para programar uma execução em um ambiente de teste padrão

- Execute schedule-run especificando o ARN do projeto, o ARN do grupo de dispositivos, o ARN de upload do aplicativo e as informações do pacote de testes.

  Exemplo:

  ```
  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
  ```

  A resposta contém um ARN de execução que você pode usar para verificar o status da execução de teste.

  ```
  {
      "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
        }
    }
}
```


Para obter mais informações, consulte schedule-run.

Para programar uma execução em um ambiente de teste personalizado

- As etapas são quase idênticas às do ambiente de teste padrão com um atributo `testSpecArn` adicional incluído no parâmetro `--test`.

  Exemplo:

```
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
```

Para verificar o status da execução de teste

- Use o comando get-run e especifique o ARN de execução.

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

Para obter mais informações, consulte get-run. Para obter informações sobre como usar o Device Farm com o AWS CLI, consulteAWS CLI referência.

Criar uma execução de teste (API)

As etapas são as mesmas descritas na AWS CLI seção. Consulte Criar uma execução de teste (AWS CLI).

Você precisa dessas informações para chamar a API `ScheduleRun`:

- Um ARN de projeto. Consulte Criar um projeto (API) e `CreateProject`.
- Um ARN de upload do aplicativo. Consulte `CreateUpload`.
- Um ARN de upload do pacote de testes. Consulte `CreateUpload`.
- ARN de um grupos de dispositivos. Consulte Criação de um pool de dispositivos e `CreateDevicePool`.

> **Note**
>
> Se estiver executando testes em um ambiente de teste personalizado, você também precisará do ARN de upload da especificação de teste. Para obter mais informações, consulte Etapa 5: (opcional) fazer upload de sua especificação de teste personalizada e `CreateUpload`.

Para obter informações sobre como usar a API do Device Farm, consulte Automatização do Device Farm.

Próximas etapas

No console do Device Farm, o ícone de relógio

muda para um ícone de resultado, como sucesso

quando a execução é concluída. Um relatório da execução é exibido assim que os testes são concluídos. Para obter mais informações, consulte Relatórios no AWS Device Farm.

Para usar o relatório, siga as instruções em Visualizando relatórios de teste no Device Farm.

Definindo o tempo limite de execução para execuções de teste no AWS Device Farm

Você pode definir por quanto tempo um teste deve ser executado antes de interromper a execução de teste de cada dispositivo. O tempo limite de execução padrão é 150 minutos por dispositivo, mas você pode definir um valor mínimo de 5 minutos. Você pode usar o console do AWS Device Farm ou a API do AWS Device Farm para definir o tempo limite de execução. AWS CLI

> **Important**
>
> A opção do tempo limite de execução deve ser definida como a duração máxima para uma execução de teste, além de algum buffer. Por exemplo, se os testes demorarem 20 minutos por dispositivo, você deverá escolher um tempo limite de 30 minutos por dispositivo.

Se a execução exceder o tempo limite, a execução nesse dispositivo será interrompida à força. Os resultados parciais estarão disponíveis, se possível. Você será cobrado pela execução até esse ponto, se estiver usando a opção de cobrança medida. Para obter mais informações sobre preços, consulte Definição de preço do AWS Device Farm.

Talvez você queira usar esse recurso se souber quanto tempo leva para executar um teste em cada dispositivo. Ao especificar um tempo limite de execução para um teste, você pode evitar a situação em que a execução fica impedida por algum motivo e você continua sendo cobrado por minutos de dispositivo mesmo quando nenhum teste está sendo executado. Em outras palavras, usar o recurso

de tempo limite de execução permite interromper essa execução se ela estiver demorando mais do que o esperado.

Você pode definir o tempo limite de execução em dois locais: no nível do projeto e no nível de execução de teste.

Pré-requisitos

1. Siga as etapas em Configuração.
2. Crie um projeto no Device Farm. Siga as instruções em Criação de um projeto no AWS Device Farm e retorne para esta página.

Definir o tempo limite de execução de um projeto

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e, em seguida, Projetos.
3. Se você já tiver um projeto, selecione-o na lista. Caso contrário, escolha Novo projeto, insira um nome para o projeto e, em seguida, selecione Enviar.
4. Escolha Configurações do projeto.
5. Na guia Geral, em Tempo limite de execução, insira um valor ou use a barra deslizante.
6. Escolha Salvar.

   Todas as execuções de teste no projeto agora usarão o valor de tempo limite de execução que você acabou de especificar, a menos que substitua o valor do tempo limite ao programar uma execução.

Definir o tempo limite de execução para uma execução de teste

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e, em seguida, Projetos.
3. Se você já tiver um projeto, selecione-o na lista. Caso contrário, escolha Novo projeto, insira um nome para o projeto e, em seguida, selecione Enviar.
4. Escolha Criar uma nova execução.

5. Siga as etapas para escolher um aplicativo, configurar o teste, selecionar os dispositivos e especificar um estado para o dispositivo.
6. Em Analisar e iniciar a execução, para Definir tempo limite de execução, insira um valor ou use a barra deslizante.
7. Escolha Confirmar e iniciar a execução.

Simulação de conexões e condições de rede para suas execuções do AWS Device Farm

Você pode usar a modelagem de rede para simular conexões e condições de rede enquanto testa suas aplicações para Android, iOS, FireOS e Web no Device Farm. Por exemplo, você pode testar o aplicativo em condições de rede inferiores às perfeitas.

Quando você cria uma execução usando as configurações de rede padrão, cada dispositivo tem uma conexão Wi-Fi completa e desimpedida com conectividade com a internet. Ao usar a modelagem de rede, você pode alterar a conexão Wi-Fi para especificar um perfil de rede, como 3G ou Lossy, WiFi que controla a taxa de transferência, o atraso, a instabilidade e a perda do tráfego de entrada e saída.

Tópicos

- Configurar a modelagem de rede ao programar uma execução de teste
- Criar um perfil de rede
- Alterar as condições da rede durante o teste

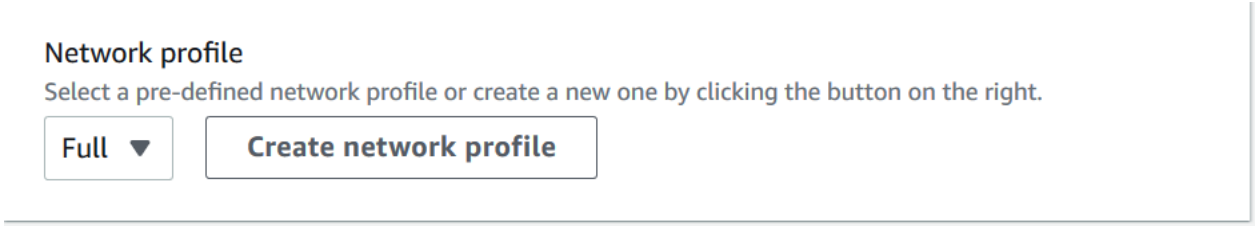
Configurar a modelagem de rede ao programar uma execução de teste

Ao programar uma execução, você pode escolher entre qualquer um dos perfis selecionados pelo Device Farm ou criar e gerenciar o seu próprio perfil.

1. Em qualquer projeto do Device Farm, escolha Criar uma nova execução.

   Se ainda não tiver um projeto, consulte Criação de um projeto no AWS Device Farm.
2. Escolha sua aplicação e, em seguida, selecione Próximo.
3. Configure seu teste e, em seguida, escolha Próximo.
4. Selecione seus dispositivos e, em seguida, escolha Próximo.

5. Na seção Configurações de localização e rede, escolha um perfil de rede ou selecione Criar perfil de rede para criar o seu próprio perfil.

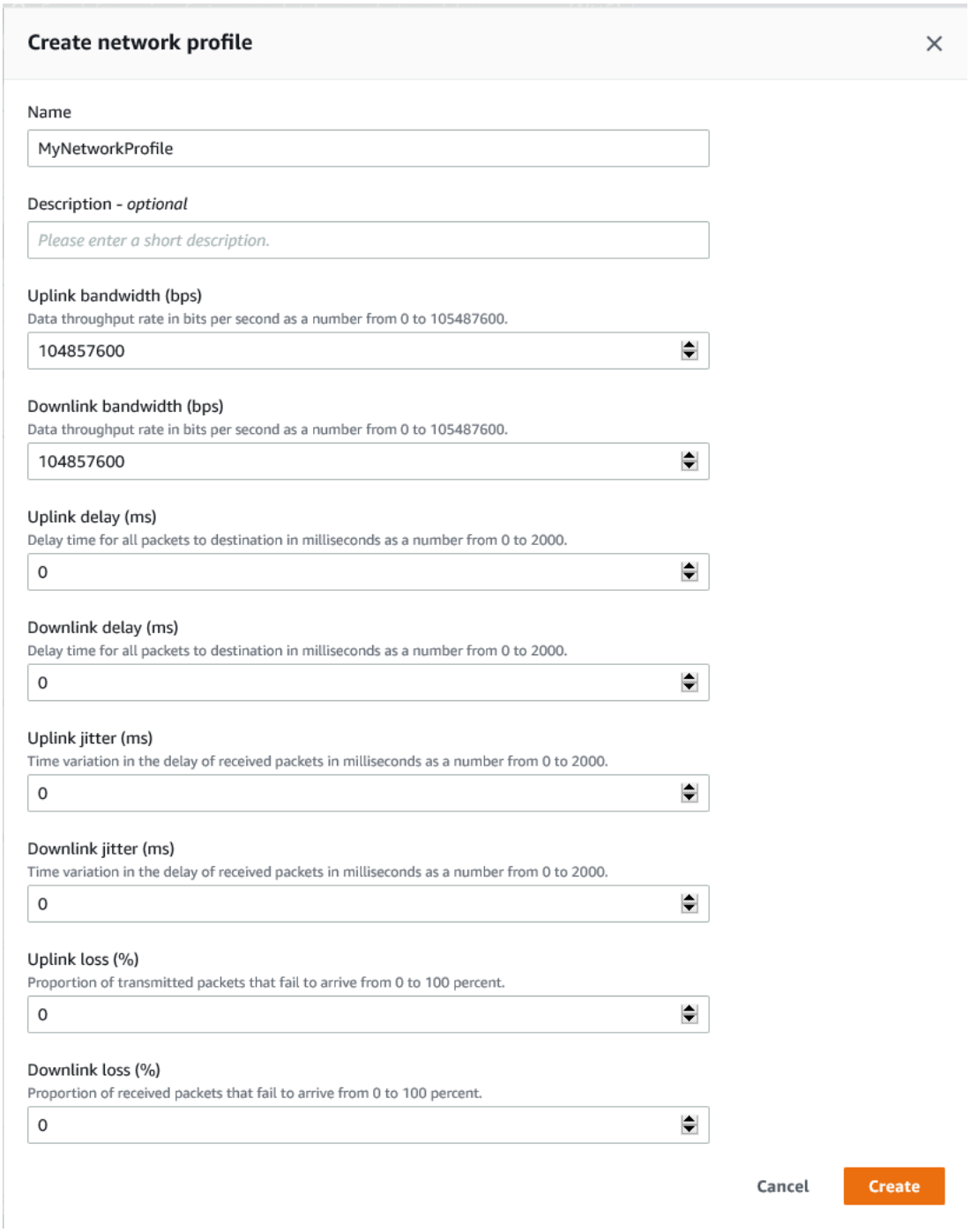


6. Escolha Próximo.
7. Analise e inicie a execução de teste.

Criar um perfil de rede

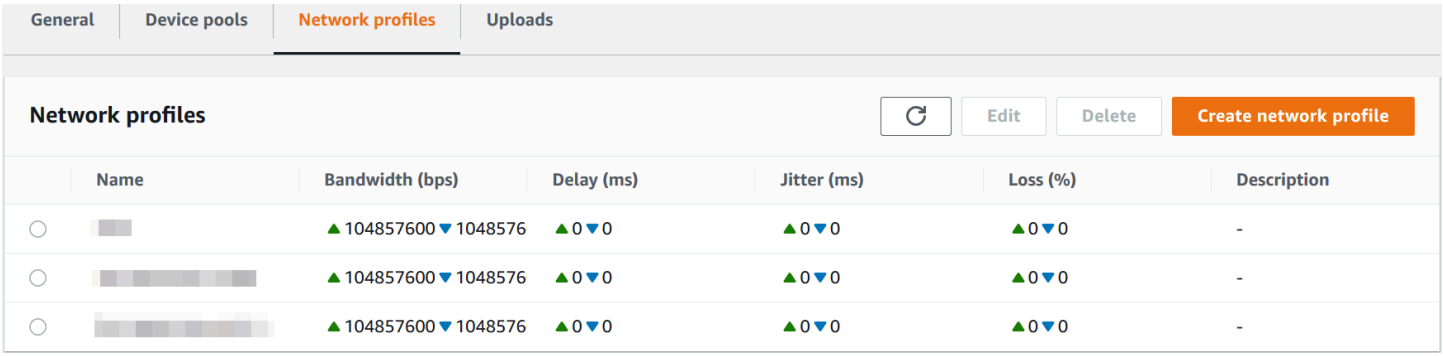
Ao criar uma execução de teste, você pode criar um perfil de rede.

1. Escolha Criar perfil de rede.



2. Digite um nome e as configurações para o perfil de rede.
3. Escolha Criar.
4. Termine de criar a execução de teste e inicie a execução.

Depois de criar um perfil de rede, você poderá vê-lo e gerenciá-lo na página Configurações do projeto.



Alterar as condições da rede durante o teste

Você pode chamar uma API pelo host do dispositivo usando uma estrutura como Appium para simular condições de rede dinâmicas, como largura de banda reduzida durante a execução do teste. Para obter mais informações, consulte CreateNetworkProfile.

Interrompendo uma execução no AWS Device Farm

Talvez você queira interromper uma execução já iniciada. Por exemplo, se perceber um problema enquanto os testes estiverem sendo executados, convém reiniciar a execução com um script de teste atualizado.

Você pode usar o console Device Farm ou a API para interromper uma execução. AWS CLI

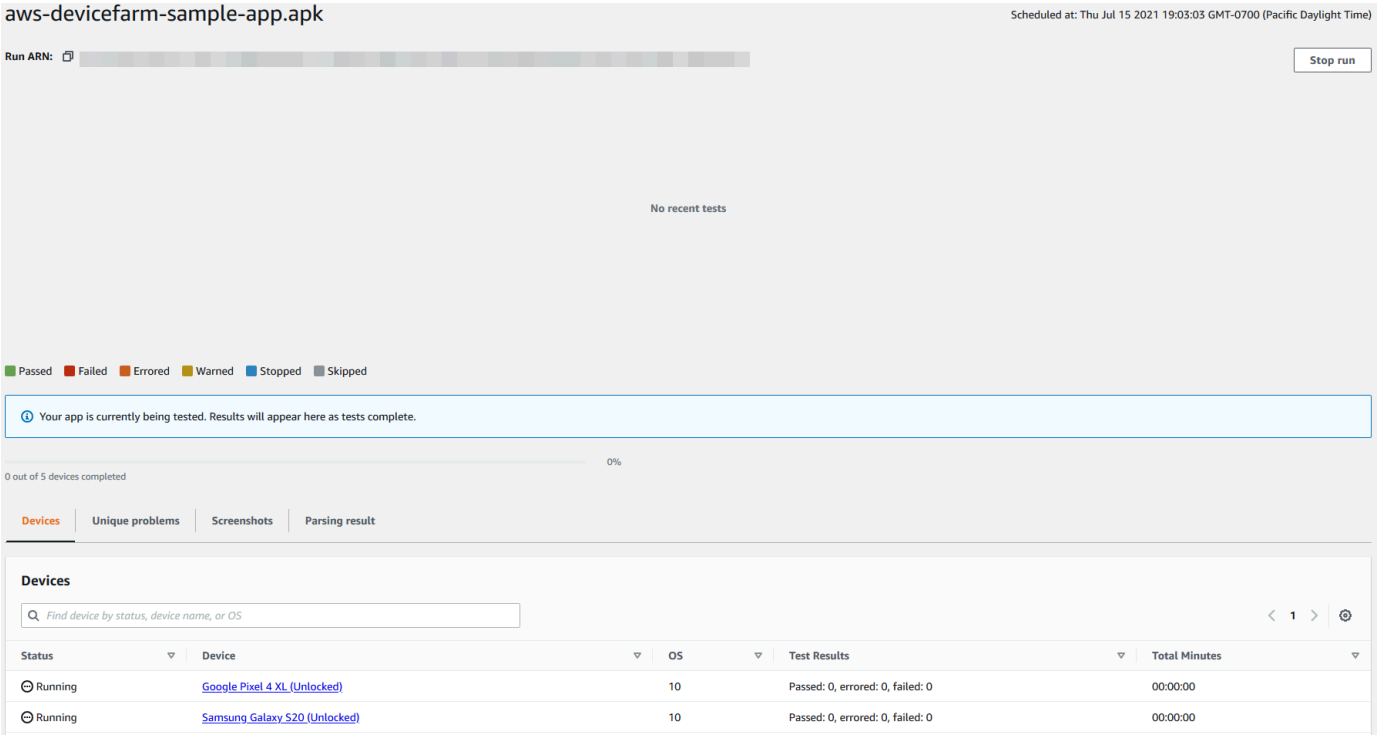
Tópicos

- Interromper uma execução (console)
- Interromper uma execução (AWS CLI)
- Interromper uma execução (API)

Interromper uma execução (console)

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e, em seguida, Projetos.
3. Escolha o projeto no qual você tem uma execução de teste ativa.
4. Na página Testes automatizados, escolha a execução do teste.

O ícone pendente ou em execução deve aparecer à esquerda do nome do dispositivo.

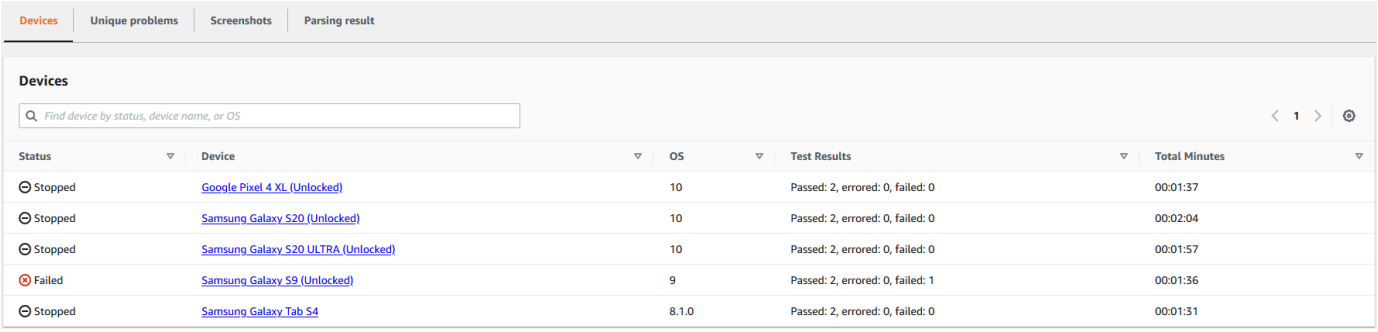


5. Escolha Interromper a execução.

Após um breve período, um ícone com um círculo vermelho com um sinal de menos dentro aparece ao lado do nome do dispositivo. Quando a execução é interrompida, a cor do ícone muda de vermelho para preto.

> **Important**
>
> Se um teste já tiver sido executado, o Device Farm não poderá interrompê-lo. Se um teste estiver em andamento, o Device Farm interromperá o teste. O total de minutos pelos quais você será cobrado é exibido na seção Dispositivos. Além disso, você também será cobrado pelo total de minutos que o Device Farm leva para executar o conjunto de configuração e o conjunto de desmontagem. Para obter mais informações, consulte Definição de preço do Device Farm.

A imagem a seguir mostra um exemplo da seção Dispositivos depois que uma execução de teste foi interrompida com êxito.



Interromper uma execução (AWS CLI)

Você pode executar o comando a seguir para interromper a execução do teste especificada, onde *myARN* está o Amazon Resource Name (ARN) da execução do teste.

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

Você deve ver uma saída semelhante a:

```
{
    "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
        }
    }
}
```

Para obter o ARN de sua execução, use o comando `list-runs`. A saída deve ser semelhante ao seguinte:

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

Para obter informações sobre como usar o Device Farm com o AWS CLI, consulteAWS CLI referência.

Interromper uma execução (API)

- Chame a StopRunoperação para a execução do teste.

Para obter informações sobre como usar a API do Device Farm, consulte Automatização do Device Farm.

Visualizando uma lista de execuções no AWS Device Farm

Você pode usar o console ou a API do Device Farm para ver uma lista de execuções de um projeto. AWS CLI

Tópicos

- Visualizar uma lista de execuções (console)
- Visualizar uma lista de execuções (AWS CLI)
- Visualizar uma lista de execuções (API)

Visualizar uma lista de execuções (console)

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e, em seguida, Projetos.
3. Na lista de projetos, escolha o projeto que corresponde à lista que você deseja visualizar.

> **Tip**
>
> Você pode usar a barra de pesquisa para filtrar a lista de projetos por nome.

Visualizar uma lista de execuções (AWS CLI)

- Execute o comando list-runs.

  Para visualizar informações sobre uma única execução, execute o comando get-run.

Para obter informações sobre como usar o Device Farm com o AWS CLI, consulteAWS CLI referência.

Visualizar uma lista de execuções (API)

- Chame a API `ListRuns`.

  Para visualizar informações sobre uma única execução, chame a API `GetRun`.

Para obter informações sobre a API do Device Farm, consulte Automatização do Device Farm.

Criação de um pool de dispositivos no AWS Device Farm

Você pode usar o console Device Farm ou a API para criar um pool de dispositivos. AWS CLI

Tópicos

- Pré-requisitos
- Criar um grupo de dispositivos (console)
- Criar um grupo de dispositivos (AWS CLI)
- Criar um grupo de dispositivos (API)

Pré-requisitos

- Crie uma execução no console do Device Farm. Siga as instruções em Criando uma execução de teste no Device Farm. Ao acessar a página Selecionar dispositivos, avance às instruções nesta seção.

Criar um grupo de dispositivos (console)

1. Na página Projetos, escolha seu projeto. Na página de detalhes do projeto, escolha Configurações do projeto. Na guia Conjuntos de dispositivos, escolha Criar pool de dispositivos.
2. Em Nome, insira um nome que facilite a identificação desse grupo de dispositivos.
3. Em Descrição, digite uma descrição que facilite a identificação desse grupo de dispositivos.
4. Se quiser usar um ou mais critérios de seleção para os dispositivos nesse grupo de dispositivos, faça o seguinte:

a. Escolha Criar grupos dinâmicos de dispositivos.

b. Escolha Adicionar uma regra.

c. Em Campo (primeira lista suspensa), escolha uma das seguintes opções:

- Para incluir dispositivos pelo nome do fabricante, escolha Fabricante do dispositivo.
- Para incluir dispositivos pelo formato (tablet ou telefone), escolha Fator de forma.
- Para incluir dispositivos por seu status de disponibilidade com base na carga, escolha Disponibilidade.
- Para incluir somente dispositivos públicos ou privados, escolha Tipo de frota.
- Para incluir dispositivos pelo sistema operacional, escolha Plataforma.
- Alguns dispositivos têm uma etiqueta ou descrição adicional sobre o dispositivo. Você pode encontrar dispositivos com base no conteúdo dos rótulos escolhendo rótulos de instância.
- Para incluir dispositivos pela versão do sistema operacional, escolha Versão do sistema operacional.
- Para incluir dispositivos por modelo, escolha Modelo.

d. Para Operador (segunda lista suspensa), escolha uma operação lógica (EQUALS, CONTAINS, etc.) para incluir dispositivos com base na consulta. Por exemplo, você pode *`Availability EQUALS AVAILABLE`* optar por incluir dispositivos que atualmente têm o `Available` status.

e. Em Valor (terceira lista suspensa), insira ou selecione o valor que deseja especificar para os valores de Campo e Operador. Os valores são limitados com base na sua escolha de campo. Por exemplo, se você escolher Plataforma para campo, as únicas seleções disponíveis serão ANDROID e IOS. Da mesma forma, se você escolher Fator forma para Campo, as únicas seleções disponíveis serão TELEFONE e TABLET.

f. Para adicionar outra regra, escolha Adicionar uma regra.

Depois que você criar a primeira regra, na lista de dispositivos, a caixa ao lado de cada dispositivo correspondente à regra será marcada. Depois que você criar ou alterar regras existentes, na lista de dispositivos, a caixa de seleção ao lado de cada dispositivo correspondente a essas regras combinadas será marcada. Os dispositivos com caixas selecionadas são incluídos no grupo de dispositivos. Os dispositivos com caixas desmarcadas são excluídos.

g. Em Máximo de dispositivos, insira o número de dispositivos que você deseja usar no seu pool de dispositivos. Se você não inserir o número máximo de dispositivos, o Device Farm escolherá todos os dispositivos da frota que correspondam às regras que você criou. Para evitar cobranças adicionais, defina esse número para um valor que corresponda aos requisitos reais de execução paralela e variedade de dispositivos.

h. Para excluir uma regra, escolha Remover regra.

5. Se quiser incluir ou excluir manualmente dispositivos individuais, faça o seguinte:

   a. Escolha Criar grupos estáticos de dispositivos.

   b. Selecione ou desmarque a caixa ao lado de cada dispositivo. Você poderá marcar ou desmarcar as caixas somente se não houver regras especificadas.

6. Se desejar incluir ou excluir todos os dispositivos exibidos, marque ou desmarque a caixa na linha de cabeçalho de coluna da lista. Se você quiser visualizar somente instâncias de dispositivos privados, escolha Ver somente instâncias de dispositivos privados.

   > **Important**
   >
   > Embora você possa usar as caixas na linha de cabeçalho da coluna para alterar a lista de dispositivos exibidos, isso não significa que os demais dispositivos exibidos sejam os únicos incluídos ou excluídos. Para confirmar quais dispositivos são incluídos ou excluídos, não se esqueça de apagar o conteúdo de todas as caixas na linha de cabeçalho da coluna e navegue nas caixas.

7. Escolha Criar.

Criar um grupo de dispositivos (AWS CLI)

> **Tip**
>
> Se você não inserir o número máximo de dispositivos, o Device Farm escolherá todos os dispositivos da frota que correspondam às regras que você criou. Para evitar cobranças adicionais, defina esse número para um valor que corresponda aos requisitos reais de execução paralela e variedade de dispositivos.

- Execute o comando create-device-pool.

Para obter informações sobre como usar o Device Farm com o AWS CLI, consulteAWS CLI referência.

Criar um grupo de dispositivos (API)

> **Tip**
>
> Se você não inserir o número máximo de dispositivos, o Device Farm escolherá todos os dispositivos da frota que correspondam às regras que você criou. Para evitar cobranças adicionais, defina esse número para um valor que corresponda aos requisitos reais de execução paralela e variedade de dispositivos.

- Chame a API `CreateDevicePool`.

Para obter informações sobre como usar a API do Device Farm, consulte Automatização do Device Farm.

Análise de resultados de testes no AWS Device Farm

No ambiente de teste padrão, é possível usar o console do Device Farm para visualizar os relatórios de cada teste na execução do teste. A visualização dos relatórios ajuda você a entender quais testes foram aprovados ou falharam e fornece detalhes sobre o desempenho e o comportamento do seu aplicativo em diferentes configurações de dispositivos.

O Device Farm também reúne outros artefatos, como arquivos, logs e imagens, que podem ser baixados quando a execução do teste for concluída. Essas informações podem ajudar você a analisar como seu aplicativo está se comportando em dispositivos reais, identificar problemas ou bugs e diagnosticar problemas.

Tópicos

- Visualizando relatórios de teste no Device Farm
- Baixando artefatos no Device Farm

Visualizando relatórios de teste no Device Farm

Use o console do Device Farm para visualizar seus relatórios de testes. Para obter mais informações, consulte Relatórios no AWS Device Farm.

Tópicos

- Pré-requisitos
- Exibir relatórios
- Status dos resultados dos testes do Device Farm

Pré-requisitos

Configure uma execução de teste e verifique se ela foi concluída.

1. Para criar uma execução, consulte Criando uma execução de teste no Device Farm e retorne a esta página.
2. Verifique se a execução foi concluída. Durante a execução do teste, o console do Device Farm exibe um ícone pendente

   para execuções que estão em andamento. Cada dispositivo na execução também começará com o ícone pendente e, em seguida, mudará para o ícone em execução

   quando o teste começar. Quando cada teste é concluído, um ícone de resultado do teste é exibido ao lado do nome do dispositivo. Quando todos os testes tiverem sido concluídos, o ícone pendente ao lado da execução mudará para um ícone de resultado de teste. Para obter mais informações, consulte Status dos resultados dos testes do Device Farm .

Exibir relatórios

Você pode visualizar os resultados do seu teste no console do Device Farm.

Tópicos

- Visualizar a página de resumo da execução de teste
- Visualizar relatórios de problemas exclusivos
- Visualizar relatórios do dispositivo

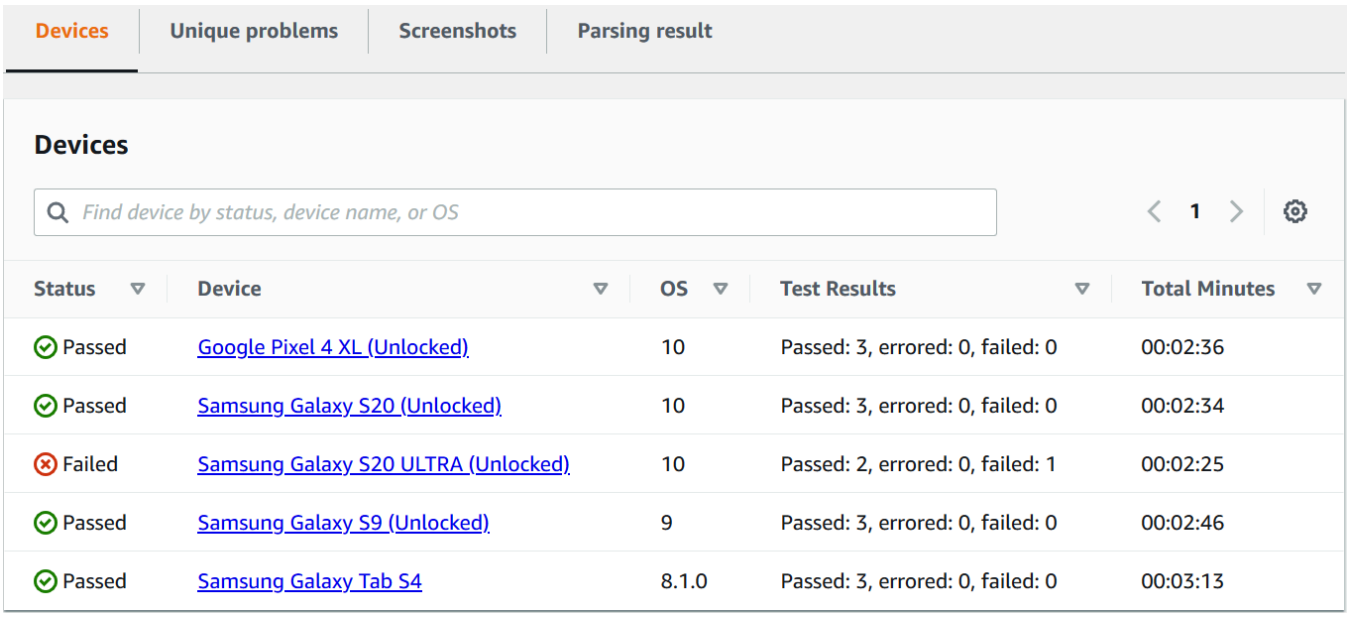
- Visualizar relatórios do conjunto de testes
- Visualizar relatórios de teste
- Visualizar dados de desempenho de um problema, dispositivo, conjunto ou teste em um relatório
- Visualizar informações de log de um problema, dispositivo, conjunto ou teste em um relatório

Visualizar a página de resumo da execução de teste

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. No painel de navegação, escolha Teste para dispositivos móveis e Projetos.
3. Na lista de projetos, escolha o projeto para a execução.

> **Tip**
>
> Para filtrar a lista de projetos por nome, use a barra de pesquisa.

4. Escolha uma execução concluída para visualizar a página de relatório resumido.
5. A página de resumo da execução de teste exibe uma visão geral dos resultados do teste.
   - A seção Problemas exclusivos lista avisos e falhas exclusivos. Para visualizar problemas exclusivos, siga as instruções em Visualizar relatórios de problemas exclusivos.
   - A seção Dispositivos exibe o número total de testes, por resultado, para cada dispositivo.



Neste exemplo, há vários dispositivos. Na primeira entrada da tabela, o dispositivo Google Pixel 4 XL com Android versão 10 relata três testes bem-sucedidos que levaram 2:36 minutos para serem executados.

Para visualizar os resultados por dispositivo, siga as instruções em Visualizar relatórios do dispositivo.

- A seção Capturas de tela exibe uma lista de todas as capturas de tela que o Device Farm fez durante a execução, agrupadas por dispositivo.
- Na seção Resultado da análise, você pode baixar o resultado da análise.

Visualizar relatórios de problemas exclusivos

1. Em Problemas exclusivos, escolha o problema que você deseja visualizar.
2. Escolha o dispositivo. O relatório exibe informações sobre o problema.

   A seção Vídeo exibe a gravação em vídeo do teste disponível para download.

   A seção Resultado exibe o resultado do teste. O status é representado como um ícone de resultado. Para obter mais informações, consulte Status de um teste individual.

   A seção Logs exibe todas as informações que o Device Farm registrou durante o teste. Para visualizar essas informações, siga as instruções em Visualizar informações de log de um problema, dispositivo, conjunto ou teste em um relatório.

   A guia Desempenho exibe informações sobre todos os dados de desempenho gerados pelo Device Farm durante o teste. Para visualizar esses dados de desempenho, siga as instruções em Visualizar dados de desempenho de um problema, dispositivo, conjunto ou teste em um relatório.

   A guia Arquivos exibe uma lista de todos os arquivos associados do teste (como arquivos de log) disponíveis para download. Para fazer download de um arquivo, selecione o link do arquivo na lista.

   A guia Capturas de tela exibe uma lista de todas as capturas de tela que o Device Farm fez durante o teste.

Visualizar relatórios do dispositivo

- Na seção Dispositivos, escolha o dispositivo.

  A seção Vídeo exibe a gravação em vídeo do teste disponível para download.

  A seção Conjuntos exibe uma tabela com informações sobre os conjuntos do dispositivo.

  Nessa tabela, a coluna Resultados do teste resume o número de testes por resultado para cada um dos conjuntos de testes que foram executados no dispositivo. Esses dados também têm um componente gráfico. Para obter mais informações, consulte Status de vários testes.

  Para visualizar os resultados completos por conjunto, siga as instruções em Visualizar relatórios do conjunto de testes.

  A seção Logs exibe todas as informações que o Device Farm registrou para o dispositivo durante a execução. Para visualizar essas informações, siga as instruções em Visualizar informações de log de um problema, dispositivo, conjunto ou teste em um relatório.

  A seção Desempenho exibe informações sobre todos os dados de desempenho que o Device Farm gerou para o dispositivo durante a execução. Para visualizar esses dados de desempenho, siga as instruções em Visualizar dados de desempenho de um problema, dispositivo, conjunto ou teste em um relatório.

  A seção Arquivos exibe uma lista de conjuntos para o dispositivo e todos os arquivos associados (como arquivos de log) que podem ser baixados. Para fazer download de um arquivo, selecione o link do arquivo na lista.

  A seção Capturas de tela exibe uma lista de todas as capturas de tela que o Device Farm fez durante a execução do dispositivo, agrupadas por conjunto.

Visualizar relatórios do conjunto de testes

1. Na seção Dispositivos, escolha o dispositivo.
2. Na seção Conjuntos, escolha o conjunto na tabela.

   A seção Vídeo exibe a gravação em vídeo do teste disponível para download.

   A seção Testes exibe uma tabela contendo informações sobre os testes no conjunto.

Na tabela, a coluna Resultados do teste exibe o resultado. Esses dados também têm um componente gráfico. Para obter mais informações, consulte Status de vários testes.

Para visualizar os resultados completos por teste, siga as instruções em Visualizar relatórios de teste.

A seção Logs exibe todas as informações que o Device Farm registrou durante a execução do conjunto. Para visualizar essas informações, siga as instruções em Visualizar informações de log de um problema, dispositivo, conjunto ou teste em um relatório.

A seção Desempenho exibe informações sobre dados de desempenho que o Device Farm gerou durante a execução do conjunto. Para visualizar esses dados de desempenho, siga as instruções em Visualizar dados de desempenho de um problema, dispositivo, conjunto ou teste em um relatório.

A seção Arquivos exibe uma lista de testes para o conjunto e todos os arquivos associados (como arquivos de log) que podem ser baixados. Para fazer download de um arquivo, selecione o link do arquivo na lista.

A seção Capturas de tela exibe uma lista de todas as capturas de tela que o Device Farm fez durante a execução do conjunto, agrupadas por teste.

Visualizar relatórios de teste

1. Na seção Dispositivos, escolha o dispositivo.
2. Na seção Conjuntos, escolha o conjunto.
3. Na seção Testes, escolha o teste.
4. A seção Vídeo exibe a gravação em vídeo do teste disponível para download.

   A seção Resultado exibe o resultado do teste. O status é representado como um ícone de resultado. Para obter mais informações, consulte Status de um teste individual.

   A seção Logs exibe todas as informações que o Device Farm registrou durante o teste. Para visualizar essas informações, siga as instruções em Visualizar informações de log de um problema, dispositivo, conjunto ou teste em um relatório.

   A guia Desempenho exibe informações sobre todos os dados de desempenho gerados pelo Device Farm durante o teste. Para visualizar esses dados de desempenho, siga as instruções

em Visualizar dados de desempenho de um problema, dispositivo, conjunto ou teste em um relatório.

A guia Arquivos exibe uma lista de todos os arquivos associados do teste (como arquivos de log) disponíveis para download. Para fazer download de um arquivo, selecione o link do arquivo na lista.

A guia Capturas de tela exibe uma lista de todas as capturas de tela que o Device Farm fez durante o teste.

Visualizar dados de desempenho de um problema, dispositivo, conjunto ou teste em um relatório

> **Note**
>
> O Device Farm coleta dados de desempenho do dispositivo somente para hosts de teste Android legados que não usam o host de `amazon_linux_2` teste mais recente. Esse recurso não é compatível com iOS.

A guia Desempenho exibe as seguintes informações:

- O grafo de CPU exibe a porcentagem de CPU que a aplicação usou em um único núcleo durante o problema, dispositivo, conjunto ou teste selecionado (ao longo do eixo vertical) ao longo do tempo (ao longo do eixo horizontal).

  O eixo vertical é expresso em porcentagem, de 0% à porcentagem registrada máxima.

  Essa porcentagem pode exceder 100% caso o aplicativo tenha usado mais de um núcleo. Por exemplo, caso três núcleos apresentem um uso de 60%, essa porcentagem é exibida como 180%.
- O grafo de Memória exibe o número de MB que a aplicação usou durante o problema, dispositivo, conjunto ou teste selecionado (no eixo vertical) ao longo do tempo (no eixo horizontal).

  O eixo vertical é expresso em MB, de 0 MB ao número máximo de MB registrado.
- O gráfico Threads exibe o número de threads usado pelo aplicativo relativamente ao problema, dispositivo, pacote ou teste selecionado (ao longo do eixo vertical) e relativamente ao tempo (ao longo do eixo horizontal).

  O eixo vertical é expresso em número de threads, de zero threads até o número máximo de threads registrados.

Em todos os casos, o eixo horizontal é representado em segundos, do início ao fim da execução relativamente ao problema, dispositivo, pacote ou teste selecionado.

Para exibir informações relacionadas a um determinado ponto de dados, pause sobre o segundo desejado ao longo do eixo horizontal do gráfico desejado.

Visualizar informações de log de um problema, dispositivo, conjunto ou teste em um relatório

A seção Logs exibe as seguintes informações:

- Origem representa a origem de uma entrada de log. Os possíveis valores incluem:
  - Harness representa uma entrada de log que o Device Farm criou. Essas entradas de log normalmente são criadas durante os eventos de início e interrupção.
  - Dispositivo representa uma entrada de log que o dispositivo criou. Para Android, essas entradas de log são compatíveis com logcat. Para iOS, essas entradas de log são compatíveis com syslog.
  - Teste representa uma entrada de log criada por um teste ou pela estrutura de teste.
- Hora representa o tempo decorrido entre a primeira entrada de log e a entrada de log em questão. A hora é expressa em *MM:SS.SSS* formato, onde *M* representa minutos e *S* representa segundos.
- PID representa o identificador de processo (PID) que criou a entrada de log. Todas as entradas de log criadas por um aplicativo em um dispositivo têm o mesmo PID.
- Nível representa o nível de registro relativo à entrada de log. Por exemplo, `Logger.debug("This is a message!")` registra o Nível de `Debug`. Estes são os valores possíveis:
  - Alerta
  - Crítico
  - Depure
  - Emergência
  - Erro
  - Com erro
  - Com falha
  - Informações
  - Interno
  - Aviso
  - Aprovada

  - Skipped
  - Interrompido
  - Detalhado
  - Avisado
  - Aviso
- Tag representa metadados arbitrários relativos à entrada de log. Por exemplo, o logcat para Android pode usar esse recurso para descrever qual parte do sistema criou a entrada de log (por exemplo, `ActivityManager`).
- Mensagem representa a mensagem ou os dados relativos à entrada de log. Por exemplo, `Logger.debug("Hello, World!")` registra uma Mensagem de `"Hello, World!"`.

Para exibir apenas uma parte das informações:

- Para mostrar todas as entradas de log que correspondem a um valor de uma coluna específica, insira o valor na barra de pesquisa. Por exemplo, para mostrar todas as entradas de log com o valor Origem de `Harness`, insira **`Harness`** na barra de pesquisa.
- Para remover todos os caracteres de uma caixa de cabeçalho de coluna, escolha o X na caixa de cabeçalho de coluna. Remover todos os caracteres de uma caixa de cabeçalho de coluna é o mesmo que inserir ***** nessa caixa de cabeçalho de coluna.

Para fazer download de todas as informações de registro do dispositivo, incluindo todos os conjuntos e testes executados, selecione Baixar logs.

Status dos resultados dos testes do Device Farm

O console do Device Farm exibe ícones que ajudam a avaliar rapidamente o estado da execução de teste concluída. Para obter mais informações sobre testes no Device Farm, consulteRelatórios no AWS Device Farm.

Tópicos

- Status de um teste individual
- Status de vários testes

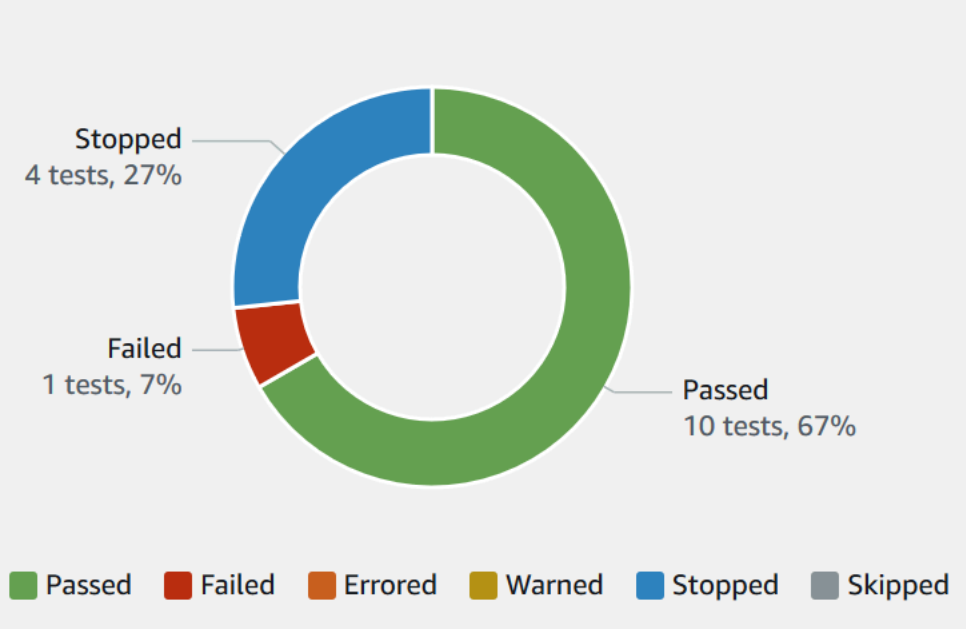
Status de um teste individual

Para relatórios que descrevem um teste individual, o Device Farm exibe um ícone representando o status do resultado do teste:

Descrição	Ícone
O teste bem-sucedido.	
Falha no teste.	
O Device Farm ignorou o teste.	
O teste foi interrompido.	
O Device Farm retornou um aviso.	
O Device Farm retornou um erro.	

Status de vários testes

Se você escolher uma execução finalizada, o Device Farm exibirá um gráfico resumido mostrando a porcentagem de testes em vários estados.



Por exemplo, esse grafo de resultados de execução de teste mostra que a execução teve 4 testes interrompidos, 1 teste com falha e 10 testes bem-sucedidos.

Os grafos são sempre codificados por cores e rotulados.

Baixando artefatos no Device Farm

O Device Farm reúne artefatos como relatórios, arquivos de log e imagens para cada teste na execução.

Você pode fazer download dos artefatos criados durante a execução de teste:

Arquivos

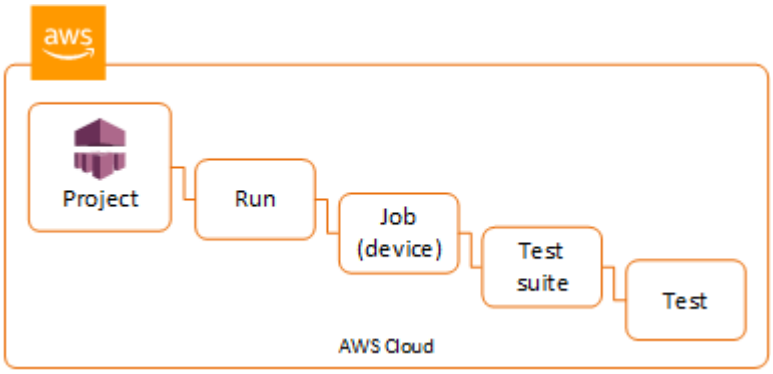
Arquivos gerados durante a execução do teste, incluindo relatórios do Device Farm. Para obter mais informações, consulte Visualizando relatórios de teste no Device Farm.

Logs

A saída de cada teste na execução.

Capturas de tela

Imagens de tela gravadas para cada teste na execução.



Baixar artefatos (console)

1. Na página de execução do relatório de teste, em Dispositivos, escolha um dispositivo móvel.
2. Para fazer download de um arquivo, escolha um dos Arquivos.
3. Para fazer download os logs da execução de teste, em Logs, escolha Fazer download de logs.

4. Para fazer download de uma captura de tela, escolha uma captura de tela Capturas de tela.

Para obter mais informações sobre como fazer download de artefatos em um ambiente de teste personalizado, consulte Baixando artefatos em um ambiente de teste personalizado.

Baixar artefatos ()AWS CLI

Você pode usar o AWS CLI para listar seus artefatos de execução de teste.

Tópicos

- Etapa 1: obter os nomes do recurso da Amazon (ARN)
- Etapa 2: listar os artefatos
- Etapa 3: fazer download dos artefatos

Etapa 1: obter os nomes do recurso da Amazon (ARN)

Você pode listar os artefatos por execução, trabalho, conjunto de testes ou teste. Você precisa do ARN correspondente. Esta tabela mostra o ARN de entrada para cada um dos comandos da AWS CLI lista:

AWS CLI Comando de lista	ARN obrigatório
list-projects	Este comando retorna todos os projetos e não exige um ARN.
list-runs	`project`
list-jobs	`run`
list-suites	`job`
list-tests	`suite`

Por exemplo, para encontrar um ARN de teste, execute list-tests usando o ARN do pacote de teste como um parâmetro de entrada.

Exemplo:

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

A resposta inclui um ARN de teste para cada um no pacote de testes.

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

Etapa 2: listar os artefatos

O comando AWS CLI list-artifacts retorna uma lista de artefatos, como arquivos, capturas de tela e registros. Cada artefato tem um URL para que você possa fazer download do arquivo.

- Chame list-artifacts especificando uma execução, um trabalho, um pacote de testes ou um ARN de teste. Especifique um tipo de ARQUIVO, LOG ou CAPTURA DE TELA.

  Este exemplo retorna um URL de download para cada artefato disponível para um teste individual:

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

A resposta contém um URL de download de cada artefato.

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

Etapa 3: fazer download dos artefatos

- Faça download do artefato usando o URL da etapa anterior. Este exemplo usa curl para fazer download de um arquivo de saída Android Appium Java:

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

Baixar artefatos (API)

O ListArtifactsmétodo Device Farm API retorna uma lista de artefatos, como arquivos, capturas de tela e registros. Cada artefato tem um URL para que você possa fazer download do arquivo.

Baixando artefatos em um ambiente de teste personalizado

Em um ambiente de teste personalizado, o Device Farm reúne artefatos como relatórios personalizados, arquivos de log e imagens. Esses artefatos estão disponíveis para cada dispositivo na execução de teste.

Você pode fazer download esses artefatos criados durante a execução de teste:

Saída da especificação de teste

A saída da execução dos comandos no arquivo YAML da especificação de teste.

Artefatos do cliente

Um arquivo compactado que contém os artefatos da execução de teste. Ele está pré-configurado na seção artifacts: do arquivo YAML da especificação de teste.

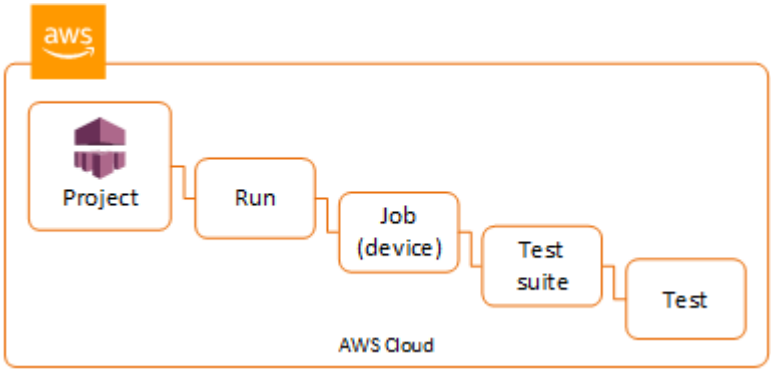
Script de shell da especificação de teste

Um arquivo de script de shell intermediário criado pelo arquivo YAML. Como é usado na execução de teste, o arquivo de script de shell pode ser usado para depurar o arquivo YAML.

Arquivo de especificação de teste

O arquivo YAML usado na execução de teste.

Para obter mais informações, consulte Baixando artefatos no Device Farm.



Marcação de recursos do AWS Device Farm

O AWS Device Farm trabalha com a API AWS Resource Groups Tagging. Essa API permite gerenciar recursos na conta da AWS com tags. É possível adicionar tags a recursos, como projetos e execuções de teste.

É possível usar tags para:

- Organizar sua conta da AWS para reproduzir sua própria estrutura de custo. Para isso, inscreva-se para obter sua conta da AWS com os valores de chave de tags incluídos. Então, para ver o custo de recursos combinados, organize suas informações de faturamento de acordo com recursos com os mesmos valores de chave de tags. Por exemplo, você pode marcar vários recursos com um nome de aplicativo, e depois organizar suas informações de faturamento para ver o custo total daquele aplicativo em vários serviços. Para obter mais informações, consulte Alocação de custos e uso de tags em Gerenciamento de faturamento e custos da AWS.
- Controlar o acesso usando políticas do IAM. Para fazer isso, crie uma política que permita o acesso a um recurso ou conjunto de recursos usando uma condição de valor de tag.
- Identificar e gerenciar execuções que tenham determinadas propriedades como tags, assim como a ramificação usada para testes.

Para obter mais informações sobre recursos de tags, consulte o whitepaper das Melhores práticas de marcação .

Tópicos

- Marcar recursos
- Pesquisa de recursos por tag
- Remover tags de recursos

Marcar recursos

A API de marcação de grupo de recursos da AWS permite adicionar, remover ou modificar tags em recursos. Para obter mais informações, consulte a Referência de API de marcação de grupo de recursos da AWS.

Para marcar um recurso, use a operação `TagResources` no endpoint `resourcegroupstaggingapi`. Essa operação usa uma lista ARNs dos serviços suportados e

uma lista de pares de valores-chave. O valor é opcional. Uma string vazia indica que não deve haver nenhum valor para essa tag. Por exemplo, o exemplo de Python a seguir marca uma série de projetos ARNs com a tag `build-config` com o valor: `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"})
```

Não é obrigatório fornecer um valor para a tag. Para definir uma tag sem valor, use uma string vazia ("") ao especificar um valor. Uma tag pode ter apenas um valor. Qualquer valor anterior da tag associado a um recurso será substituído pelo novo valor.

Pesquisa de recursos por tag

Para pesquisar recursos por suas tags, use a operação `GetResources` no endpoint `resourcegrouptaggingapi`. Essa operação usa uma série de filtros, nenhum dos quais é necessário, e retorna os recursos que correspondem aos critérios fornecidos. Sem filtros, todos os recursos marcados são retornados. A operação `GetResources` permite filtrar recursos com base em

- Valor da tag
- Tipo de recurso (por exemplo, `devicefarm:run`)

Para obter mais informações, consulte a Referência de API de marcação de grupo de recursos da AWS.

O exemplo a seguir procura as sessões de teste do navegador de desktop do Device Farm (recursos `devicefarm:testgrid-session`) com a tag `stack` que tem o valor `production`:

```
import boto3
client = boto3.client('resourcegroupstaggingapi')
```

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

Remover tags de recursos

Para remover uma tag, use a operação `UntagResources`, especificando uma lista de recursos e as tags a serem removidas:

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

Estruturas de teste e testes integrados no AWS Device Farm

Esta seção descreve o suporte do Device Farm para estruturas de teste e tipos de teste incorporados.

Para obter mais informações sobre como o Device Farm executa testes, consulteAmbientes de teste no AWS Device Farm.

Testar estruturas

O Device Farm é compatível com essas estruturas de teste de automação móvel:

Estruturas de teste de aplicações Android

- Appium
- Instrumentação

Estruturas de teste de aplicações iOS

- Appium
- XCTest
- XCTest UI

Estruturas de teste de aplicações web

Aplicativos Web são compatíveis usando o Appium. Para obter mais informações sobre como trazer seus testes para o Appium, consulte Testes Appium e AWS Device Farm.

Estruturas em um ambiente de teste personalizado

O Device Farm não fornece suporte para personalizar o ambiente de teste da XCTest estrutura. Para obter mais informações, consulte Ambientes de teste personalizados no AWS Device Farm.

Suporte da versão do Appium

Para testes executados em um ambiente personalizado, o Device Farm é compatível com o Appium versão 1. Para obter mais informações, consulte Ambientes de teste no AWS Device Farm.

Tipos de teste integrado

Com os testes integrados, você pode testar a aplicação em vários dispositivos sem precisar escrever e manter scripts de automação de testes. O Device Farm oferece um tipo de teste integrado:

- Integrado: Fuzz (Android e iOS)

Testes Appium e AWS Device Farm

Esta seção descreve como configurar, empacotar e carregar seus testes do Appium no Device Farm. O Appium é uma ferramenta de código aberto para automatizar aplicações Web nativas e móveis. Para obter mais informações sobre o Appium, consulte Introdução ao Appium no site do Appium.

Para ver um aplicativo de amostra e links para testes em funcionamento, consulte Device Farm Sample App para Android e Device Farm Sample App para iOS em GitHub.

Para obter mais informações sobre testes no Device Farm, consulteEstruturas de teste e testes integrados no AWS Device Farm.

Suporte à versão

O suporte para várias estruturas de trabalho e linguagens de programação depende da linguagem utilizada.

O Device Farm oferece suporte a todas as versões do servidor Appium 1.x e 2.x. Para Android, você pode escolher qualquer versão principal do Appium com `devicefarm-cli`. Por exemplo, para usar a versão 2 do servidor Appium, adicione esses comandos ao arquivo YAML de especificações de teste:

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

Para iOS, você pode escolher versões específicas do Appium com os comandos `avm` ou `npm`. Por exemplo, para usar o comando `avm` para definir a versão do servidor Appium como 2.1.2, adicione esses comandos ao arquivo YAML de especificações de teste:

```
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
```

Ao usar o comando npm para usar a versão mais recente do Appium 2, adicione esses comandos ao arquivo YAML de especificação de teste:

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

Para obter mais informações sobre `devicefarm-cli` ou qualquer outro comando da CLI, consulte a Referência da AWS CLI.

Para usar todos os recursos da estrutura, como anotações, escolha um ambiente de teste personalizado e use a AWS CLI ou o console do `Device Farm` para carregar uma especificação de teste personalizada.

Integrando os testes do Appium com o Device Farm

Use as instruções a seguir para integrar os testes do Appium com o AWS Device Farm. Para obter mais informações sobre o uso dos testes Appium no Device Farm, consulte. Testes Appium e AWS Device Farm

Configurar o pacote de testes do Appium

Use as instruções a seguir para configurar o pacote de testes.

Java (JUnit)

1. Modifique `pom.xml` para definir o empacotamento em um arquivo JAR:

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

```
<packaging>jar</packaging>
```

2. Modifique `pom.xml` para usar `maven-jar-plugin` a fim de criar seus testes em um arquivo JAR.

   O plug-in a seguir cria seu código-fonte de teste (qualquer coisa no diretório `src/test`) em um arquivo 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. Modifique `pom.xml` para usar `maven-dependency-plugin` a fim de criar dependências como arquivos JAR.

   O plug-in a seguir copia suas dependências no diretório `dependency-jars`:

```
<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. Salve a montagem XML a seguir em `src/main/assembly/zip.xml`.

   O XML a seguir é uma definição de montagem que, quando configurada, instrui o Maven a criar um arquivo .zip que contém tudo o que está na raiz do diretório de saída da compilação e no diretório `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. Modifique `pom.xml` para usar `maven-assembly-plugin` a fim de empacotar testes e todas as dependências em um único arquivo .zip.

   O plug-in a seguir usa a montagem anterior para criar um arquivo .zip denominado `zip-with-dependencies` no diretório de saída da compilação toda vez que o comando mvn package for executado:

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

> **Note**
>
> Se receber uma mensagem de erro informando que a versão 1.3 não oferece suporte a anotações, adicione o seguinte ao `pom.xml`:
>
> ```
> <plugin>
>   <artifactId>maven-compiler-plugin</artifactId>
>   <configuration>
>     <source>1.7</source>
>     <target>1.7</target>
>   </configuration>
> </plugin>
> ```

Java (TestNG)

1. Modifique `pom.xml` para definir o empacotamento em um arquivo JAR:

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

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

2. Modifique `pom.xml` para usar `maven-jar-plugin` a fim de criar seus testes em um arquivo JAR.

   O plug-in a seguir cria seu código-fonte de teste (qualquer coisa no diretório `src/test`) em um arquivo 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. Modifique `pom.xml` para usar `maven-dependency-plugin` a fim de criar dependências como arquivos JAR.

   O plug-in a seguir copia suas dependências no diretório `dependency-jars`:

```
<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. Salve a montagem XML a seguir em `src/main/assembly/zip.xml`.

   O XML a seguir é uma definição de montagem que, quando configurada, instrui o Maven a criar um arquivo .zip que contém tudo o que está na raiz do diretório de saída da compilação e no diretório `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. Modifique `pom.xml` para usar `maven-assembly-plugin` a fim de empacotar testes e todas as dependências em um único arquivo .zip.

O plug-in a seguir usa a montagem anterior para criar um arquivo .zip denominado `zip-with-dependencies` no diretório de saída da compilação toda vez que o comando mvn package for executado:

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

Note

Se receber uma mensagem de erro informando que a versão 1.3 não oferece suporte a anotações, adicione o seguinte ao `pom.xml`:

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

Node.JS

Para empacotar seus testes do Appium Node.js e carregá-los no Device Farm, você deve instalar o seguinte em seu computador local:

- Gerenciador de versão do Node (nvm)

  Use essa ferramenta ao desenvolver e empacotar seus testes para que dependências desnecessárias não sejam incluídas no pacote de testes.
- Node.js
- npm-bundle (instalado globalmente)

1. Verifique se o nvm está presente

   ```
   command -v nvm
   ```

   Você verá `nvm` como resultado.

   Para obter mais informações, consulte nvm on. GitHub
2. Execute este comando para instalar o Node.js:

   ```
   nvm install node
   ```

   É possível especificar uma determinada versão do Node.js:

   ```
   nvm install 11.4.0
   ```
3. Verifique se a versão correta do Node está em uso:

   ```
   node -v
   ```
4. Instale npm-bundle globalmente:

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

Python

1. É altamente recomendável configurar o virtualenv do Python para testes de desenvolvimento e empacotamento, para que dependências desnecessárias não sejam incluídas em seu pacote de aplicativos.

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

> **Tip**
>
> - Não crie um virtualenv do Python com a opção `--system-site-packages`, porque ele herda pacotes do diretório global de pacotes de sites. Isso pode resultar na inclusão de dependências em seu ambiente virtual que não são necessárias aos seus testes.
> - Você também deve verificar se os testes não usam dependências que dependem de bibliotecas nativas, pois essas bibliotecas nativas podem ou não estar presentes na instância em que esses testes são executados.

2. Instale o py.test em seu ambiente virtual.

```
$ pip install pytest
```

3. Instale o cliente do Appium Python em seu ambiente virtual.

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

4. A menos que você especifique um caminho diferente no modo personalizado, o Device Farm espera que seus testes sejam armazenados em `tests/`. É possível usar `find` para mostrar todos os arquivos dentro de uma pasta:

```
$ find tests/
```

Confirmar se esses arquivos contêm conjuntos de testes que você deseja executar no Device Farm

```
tests/
```

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

5. Execute esse comando na pasta workspace de seu ambiente virtual para exibir uma lista de seus testes sem executá-los.

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

   Confirme se a saída mostra os testes que você deseja executar no Device Farm.

6. Limpe todos os arquivos em cache dos testes ou da pasta:

```
$ 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. Execute o comando a seguir no espaço de trabalho para gerar o arquivo requirements.txt:

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

Ruby

Para empacotar seus testes Ruby do Appium e carregá-los no Device Farm, você deve instalar o seguinte em seu computador local:

- Gerenciador de versão do Ruby (RVM)

  Use essa ferramenta de linha de comando ao desenvolver e empacotar seus testes para que dependências desnecessárias não sejam incluídas no pacote de testes.
- Ruby
- Bundler (esse gem normalmente é instalado com o Ruby.)

1. Instale as chaves obrigatórias, o RVM e o Ruby. Para obter instruções, consulte Como instalar o RVM no site do RVM.

   Depois que a instalação for concluída, atualize o terminal. Para isso, saia e faça login novamente.

> **Note**
>
> O RVM é carregado como uma função apenas para o shell bash.

2. Verifique se o rvm está instalado corretamente

```
command -v rvm
```

Você verá `rvm` como resultado.

3. Se você quiser instalar uma versão específica do Ruby, como, por exemplo*2.5.3*, execute o seguinte comando:

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

Verifique se você está na versão solicitada do Ruby:

```
ruby -v
```

4. Configure o empacotador para compilar pacotes para as plataformas de teste desejadas:

```
bundle config specific_platform true
```

5. Atualize seu arquivo .lock para adicionar as plataformas necessárias para executar os testes.

   - Se estiver compilando testes para serem executados em dispositivos Android, execute este comando para configurar o Gemfile para usar as dependências do host de teste do Android:

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

   - Se estiver compilando testes para serem executados em dispositivos iOS, execute este comando para configurar o Gemfile para usar as dependências do host de teste do iOS:

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

6. O gem bundler geralmente é instalado por padrão. Se não estiver, instale-o:

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

Criar um arquivo de pacote de teste compactado

> **Warning**
>
> No Device Farm, a estrutura de pastas dos arquivos em seu pacote de teste compactado é importante, e algumas ferramentas de arquivamento alterarão a estrutura do seu arquivo ZIP implicitamente. Recomendamos que você siga os utilitários da linha de comando especificados abaixo, em vez de usar os utilitários de arquivamento incorporados ao gerenciador de arquivos da área de trabalho local (como o Finder ou o Windows Explorer).

Agora, empacote seus testes para o Device Farm.

Java (JUnit)

Crie e empacote seus testes:

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

O arquivo `zip-with-dependencies.zip` será criado como resultado. Esse é o seu pacote de testes.

Java (TestNG)

Crie e empacote seus testes:

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

O arquivo `zip-with-dependencies.zip` será criado como resultado. Esse é o seu pacote de testes.

Node.JS

1. Confira o projeto.

   Verifique se você está no diretório raiz do seu projeto. Você pode ver `package.json` no diretório raiz.

2. Execute este comando para instalar suas dependências locais.

   ```
   npm install
   ```

Esse comando também cria uma pasta `node_modules` dentro do seu diretório atual.

> **Note**
>
> Então, você poderá executar seus testes localmente.

3. Execute este comando para empacotar os arquivos da pasta atual em um arquivo *.tgz. O arquivo recebe um nome por meio da propriedade `name` no arquivo `package.json`.

```
npm-bundle
```

Esse arquivo tarball (.tgz) contém todos os seus códigos e dependências.

4. Execute este comando para empacotar o tarball (arquivo *.tgz) gerado na etapa anterior em um único arquivo compactado:

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

Esse é o arquivo `MyTests.zip` que você carrega no Device Farm no procedimento a seguir.

Python

Python 2

Gere um arquivamento dos pacotes do Python necessários (chamados de "wheelhouse") usando pip:

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

Empacote os requisitos de pip, wheelhouse e testes em um arquivamento zip para o Device Farm:

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

Python 3

Empacote os requisitos de pip e testes em um arquivo zip:

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

Ruby

1. Execute este comando para criar um ambiente virtual do Ruby:

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

2. Execute este comando para usar o ambiente que você acabou de criar:

   ```
   rvm gemset use myGemset
   ```

3. Confira o código-fonte.

   Verifique se você está no diretório raiz do seu projeto. Você pode ver `Gemfile` no diretório raiz.

4. Execute este comando para instalar suas dependências locais e todos os gems do `Gemfile`:

   ```
   bundle install
   ```

   > **Note**
   >
   > Então, você poderá executar seus testes localmente. Use este comando para executar um teste localmente:
   >
   > ```
   > bundle exec $test_command
   > ```

5. Empacote os gems na pasta `vendor/cache`.

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

6. Execute o comando a seguir para empacotar seu código-fonte, junto com todas as suas dependências, em um único arquivo compactado:

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

Esse é o arquivo `MyTests.zip` que você carrega no Device Farm no procedimento a seguir.

Fazer upload do pacote de testes no Device Farm

Você pode usar o console do Device Farm para carregar seus testes.

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e, em seguida, Projetos.
3. Se você for um novo usuário, escolha Novo projeto, insira um nome para o projeto e escolha Enviar.

   Se você já tiver um projeto, poderá selecioná-lo para carregar seus testes nele.
4. Abra seu projeto e escolha Criar execução.
5. Em Configurações de execução, dê ao teste um nome apropriado. Ele pode conter qualquer combinação de espaços ou pontuação.
6. Para testes nativos para Android e iOS

   Em Configurações de execução, escolha aplicativo Android se estiver testando um aplicativo Android (.apk) ou escolha aplicativo iOS se estiver testando um aplicativo iOS (.ipa). Em seguida, em Selecionar aplicativo, selecione Carregar seu próprio aplicativo para carregar o pacote distribuível do seu aplicativo.

   > **Note**
   >
   > O arquivo deve ser um `.apk` para Android ou um `.ipa` para iOS. Aplicativos para iOS devem ser criados para dispositivos reais, não para o Simulator.

   Para testes de aplicativos Web para dispositivos móveis

   Em Configurações de execução, escolha Aplicativo Web.
7. Em Configurar teste, na seção Selecionar estrutura de teste, escolha a estrutura Appium com a qual você testa e, em seguida, carregue seu próprio pacote de teste.

8. Procure e escolha o arquivo .zip que contém os testes. O arquivo .zip deve seguir o formato descrito em Configurar o pacote de testes do Appium.
9. Siga as instruções para selecionar dispositivos e iniciar a execução. Para obter mais informações, consulte Criando uma execução de teste no Device Farm.

> **Note**
>
> O Device Farm não modifica os testes do Appium.

Fazer capturas de tela de seus testes (opcional)

Você pode fazer capturas de tela como parte dos testes.

O Device Farm define a propriedade `DEVICEFARM_SCREENSHOT_PATH` para um caminho totalmente qualificado no sistema de arquivos local em que o Device Farm espera que as capturas de tela do Appium sejam salvas. O diretório específico de teste no qual as capturas de tela são armazenadas é definido no runtime. As capturas de tela são inseridas automaticamente nos relatórios do Device Farm. Para visualizar as capturas de tela, no console do Device Farm, selecione a seção Capturas de tela.

Para obter mais informações sobre como fazer capturas de tela em testes do Appium, consulte Fazer captura de tela na documentação da API do Appium.

Testes de Android no AWS Device Farm

O Device Farm oferece suporte a vários tipos de testes de automação para dispositivos Android e dois testes integrados.

Para obter mais informações sobre testes no Device Farm, consulteEstruturas de teste e testes integrados no AWS Device Farm.

Estruturas de teste de aplicações Android

Os testes personalizados estão disponíveis para dispositivos Android.

- Appium

- Instrumentação

Tipos de teste incorporados para Android

Há um tipo de teste integrado disponível para dispositivos Android:

- Integrado: Fuzz (Android e iOS)

Instrumentação para Android e AWS Device Farm

O Device Farm fornece suporte para Instrumentação (JUnit, Espresso, Robotium ou qualquer teste baseado em instrumentação) para Android.

O Device Farm também fornece uma aplicação Android de amostra e links para testes funcionais em três estruturas de automação do Android, incluindo Instrumentation (Espresso). O aplicativo de amostra Device Farm para Android está disponível para download em GitHub.

Para obter mais informações sobre testes no Device Farm, consulteEstruturas de teste e testes integrados no AWS Device Farm.

Tópicos

- O que é instrumentação?
- Considerações para testes de instrumentação do Android
- Análise de teste em modo padrão
- Integrando a instrumentação Android com o Device Farm

O que é instrumentação?

A instrumentação do Android possibilita chamar métodos de retorno de chamada no código de teste, de maneira que você possa percorrer o ciclo de vida de um componente passo a passo, como se estivesse depurando o componente. Para obter mais informações, consulte Instrumented tests na seção Test types and locations da documentação de Android Developer Tools.

Considerações para testes de instrumentação do Android

Ao usar a instrumentação do Android, considere as recomendações e notas a seguir.

Verifique a compatibilidade do sistema operacional Android

Verifique a documentação do Android para garantir que o Instrumentation seja compatível com sua versão do sistema operacional Android.

Executando a partir da linha de comando

Para executar testes de instrumentação na linha de comando, siga a documentação do Android.

Animações de sistema

De acordo com a Android documentation for Espresso testing, recomenda-se que as animações do sistema sejam desativadas ao testar em dispositivos reais. O Device Farm desativa automaticamente as configurações Window Animation Scale, Transition Animation Scale e Animator Duration Scale quando executado com o executor de testes de instrumentação JUnitandroid.support.test.runner.Android Runner.

Gravadores de teste

O Device Farm oferece suporte a estruturas, como Robotium, que têm record-and-playback ferramentas de script.

Análise de teste em modo padrão

No modo padrão de uma execução, o Device Farm analisa sua suíte de testes e identifica as classes e métodos de teste exclusivos que ela executará. Isso é feito por meio de uma ferramenta chamada Dex Test Parser.

Quando recebe um arquivo.apk de instrumentação do Android como entrada, o analisador retorna os nomes dos métodos totalmente qualificados dos testes que correspondem às convenções JUnit 3 e 4. JUnit

Para testar isso em um ambiente local:

1. Faça download do binário `dex-test-parser`.
2. Execute o comando a seguir para obter a lista de métodos de teste que serão executados no Device Farm:

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

Integrando a instrumentação Android com o Device Farm

> **Note**
>
> Use as instruções a seguir para integrar os testes de instrumentação do Android com o AWS Device Farm. Para obter mais informações sobre o uso de testes de instrumentação no Device Farm, consulte. Instrumentação para Android e AWS Device Farm

Upload dos testes de instrumentação para Android

Use o console do Device Farm para carregar seus testes.

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e, em seguida, Projetos.
3. Na lista de projetos, escolha o projeto para o qual deseja carregar seus testes.

   > **Tip**
   >
   > Você pode usar a barra de pesquisa para filtrar a lista de projetos por nome.
   > Para criar um projeto, siga as instruções em Criação de um projeto no AWS Device Farm.

4. Selecione Criar execução.
5. Em Selecionar aplicativo, na seção Opções de seleção de aplicativos, selecione Carregar seu próprio aplicativo.
6. Procure e escolha o arquivo de seu aplicativo Android. O arquivo deve ser .apk.
7. Em Configurar teste, na seção Selecionar estrutura de teste, escolha Instrumentação e, em seguida, selecione Escolher arquivo.
8. Procure e escolha o arquivo .apk que contém os testes.
9. Complete as instruções restantes para selecionar dispositivos e iniciar a execução.

(Opcional) Faça capturas de tela nos testes de instrumentação do Android

Você pode fazer capturas de tela como parte dos testes de instrumentação para Android.

Para fazer a captura de telas, chame um dos seguintes métodos:

- Para Robotium, chame o método `takeScreenShot` (por exemplo, `solo.takeScreenShot();`).
- Para Spoon, chame o método `screenshot`; por exemplo:

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

Durante uma execução de teste, o Device Farm obtém capturas de tela dos seguintes locais nos dispositivos, se existirem, e as adiciona aos relatórios de teste:

- `/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`*

Testes de iOS no AWS Device Farm

O Device Farm oferece suporte a vários tipos de testes de automação para dispositivos iOS e um teste incorporado.

Para obter mais informações sobre testes no Device Farm, consulteEstruturas de teste e testes integrados no AWS Device Farm.

Estruturas de teste de aplicações iOS

Os testes a seguir estão disponíveis para dispositivos iOS.

- Appium
- XCTest
- XCTest UI

Tipos de teste integrados para iOS

No momento, existe apenas um tipo de teste integrado disponível para dispositivos iOS.

- Integrado: Fuzz (Android e iOS)

Integrando o Device Farm com XCTest o iOS

Com o Device Farm, você pode usar a XCTest estrutura para testar seu aplicativo em dispositivos reais. Para obter mais informações sobre XCTest, consulte Noções básicas sobre testes com o Xcode.

Para executar um teste, você cria os pacotes para a execução do teste e faz o upload desses pacotes para o Device Farm.

Para obter mais informações sobre testes no Device Farm, consulteEstruturas de teste e testes integrados no AWS Device Farm.

Tópicos

- Crie os pacotes para sua XCTest corrida
- Faça o upload dos pacotes para sua XCTest execução no Device Farm

Crie os pacotes para sua XCTest corrida

Para testar seu aplicativo usando a XCTest estrutura, o Device Farm exige o seguinte:

- Que seu pacote de aplicativos seja um arquivo `.ipa`.
- Seu XCTest pacote como um `.zip` arquivo.

Que você crie esses pacotes usando a saída da compilação gerada pelo Xcode. Conclua as etapas a seguir para criar os pacotes para que você possa carregá-los no Device Farm.

Para gerar a saída da compilação para seu aplicativo

1. Abra o projeto do aplicativo em Xcode.
2. No menu suspenso do esquema na barra de ferramentas do Xcode, escolha Dispositivo iOS genérico como destino.
3. No menu Produto, selecione Compilar para e depois selecione Teste.

Para criar o pacote de aplicativos

1. No navegador de projeto no Xcode, em Produtos, abra o menu contextual do arquivo chamado *app-project-name*.app. Escolha Mostrar no Finder. O Finder abrirá uma pasta chamada Debug-iphoneos, que contém a saída gerada pelo Xcode para sua compilação de teste. Essa pasta inclui o arquivo .app .
2. No Finder, crie uma nova pasta e nomeie-a Payload.
3. Copie o arquivo *app-project-name*.app e cole-o na pasta Payload.
4. Abra o menu contextual da pasta Payload e escolha Compactar "Payload". Um arquivo chamado Payload.zip será criado.
5. Altere o nome do arquivo e a extensão de Payload.zip para *app-project-name*.ipa.

   Em uma etapa posterior, você fornecerá esse arquivo ao Device Farm. Para facilitar a localização do arquivo, você pode movê-lo para outro local, como sua área de trabalho.
6. Opcionalmente, você pode excluir a pasta Payload e o arquivo .app que está nela.

Para criar o XCTest pacote

1. No Finder, no diretório Debug-iphoneos, abra o menu contextual do arquivo *app-project-name*.app. Escolha Mostrar conteúdo do pacote.
2. No conteúdo do pacote, abra a pasta Plugins. Essa pasta contém um arquivo chamado *app-project-name*.xctest.
3. Abra o menu contextual desse arquivo e escolha Compactar "***app-project-name*.xctest**". Um arquivo chamado *app-project-name*.xctest.zip será criado.

   Em uma etapa posterior, você fornecerá esse arquivo ao Device Farm. Para facilitar a localização do arquivo, você pode movê-lo para outro local, como sua área de trabalho.

Faça o upload dos pacotes para sua XCTest execução no Device Farm

Use o console do Device Farm para carregar os pacotes do seu teste.

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. Se você ainda não tiver um projeto, crie um. Para obter as etapas para criar um projeto, consulte Criação de um projeto no AWS Device Farm.

Caso contrário, no painel de navegação do Device Farm, escolha Teste para dispositivos móveis e escolha Projetos.

3. Escolha o projeto que você deseja usar para executar o teste.
4. Escolha Criar execução.
5. Em Configurações de execução, na seção Tipo de execução, escolha aplicativo para iOS.
6. Em Selecionar aplicativo, na seção Opções de seleção de aplicativos, selecione Carregar seu próprio aplicativo. Em seguida, selecione Escolher arquivo em Carregar aplicativo.
7. Procure o arquivo `.ipa` para seu aplicativo e faça o upload dele.

> **Note**
>
> Seu pacote `.ipa` deve ser compilado para testes.

8. Em Configurar teste, na seção Selecionar estrutura de teste, escolha XCTest. Em seguida, selecione Escolher arquivo em Carregar aplicativo.
9. Navegue até o `.zip` arquivo que contém o XCTest pacote do seu aplicativo e faça o upload dele.
10. Conclua as etapas restantes no processo de criação do projeto. Selecione os dispositivos que deseja testar e especifique o estado do dispositivo.
11. Escolha Criar execução. O Device Farm executa o teste e mostra os resultados no console.

Integrando a XCTest interface do usuário para iOS com o Device Farm

O Device Farm fornece suporte para a estrutura de teste de XCTest interface do usuário. Especificamente, o Device Farm suporta testes de XCTest interface de usuário escritos em Objective-C e Swift.

A estrutura de XCTest interface do usuário permite o teste de interface do usuário no desenvolvimento para iOS, construído sobre XCTest o. Para obter mais informações, consulte Teste da interface do usuário na biblioteca de desenvolvedor do iOS.

Para obter informações gerais sobre testes no Device Farm, consulteEstruturas de teste e testes integrados no AWS Device Farm.

Use as instruções a seguir para integrar o Device Farm à estrutura de teste de XCTest interface do usuário para iOS.

Tópicos

- Prepare seus testes de XCTest interface do iOS
- Opção 1: criar um pacote XCTest UI .ipa
- Opção 2: criar um pacote XCTest UI .zip
- Carregue seus testes de XCTest interface do iOS

Prepare seus testes de XCTest interface do iOS

Você pode carregar um `.ipa` arquivo ou um `.zip` arquivo para seu pacote de teste XCTEST_UI.

Um `.ipa` arquivo é um arquivo de aplicativo contendo o aplicativo iOS Runner em formato de pacote. Arquivos adicionais não podem ser incluídos no *`.ipa`* arquivo.

Se você fizer upload de um `.zip` arquivo, ele poderá conter diretamente o aplicativo iOS Runner ou um `.ipa` arquivo. Você também pode incluir outros arquivos no `.zip` arquivo se quiser usá-los durante os testes. Por exemplo, você pode incluir arquivos como`.xctestrun`, `.xcworkspace` ou `.xcodeproj` dentro de um `.zip` arquivo para executar planos de teste XCUI no farm de dispositivos. Instruções detalhadas sobre como executar planos de teste estão disponíveis no arquivo de especificação de teste padrão para o tipo de teste XCUI.

Opção 1: criar um pacote XCTest UI .ipa

O pacote yourAppNameUITest-runner.app é produzido pelo Xcode quando você cria seu projeto para teste. Ele pode ser encontrado no diretório Products do projeto.

Para criar um arquivo.ipa:

1. Crie um diretório denominado *`Payload`*.
2. Adicione o diretório do seu aplicativo ao diretório Payload.
3. Arquive o diretório Payload em um `.zip` arquivo e, em seguida, altere a extensão do arquivo para`.ipa`.

A estrutura de pastas a seguir mostra como um aplicativo de exemplo chamado *`my-project-nameUITest-Runner.app`* seria empacotado como um `.ipa` arquivo:

```
.
```

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

Opção 2: criar um pacote XCTest UI .zip

O Device Farm gera automaticamente um `.xctestrun` arquivo para você executar seu conjunto completo de testes de XCTest interface do usuário. Se quiser usar seu próprio `.xctestrun` arquivo no Device Farm, você pode compactar seus `.xctestrun` arquivos e o diretório do aplicativo em um `.zip` arquivo. Se você já tem um `.ipa` arquivo para seu pacote de teste, você pode incluí-lo aqui em vez de*`*-Runner.app`*.

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

Se você quiser executar um plano de teste do Xcode para seus testes XCUI no Device Farm, você pode criar um zip contendo seu arquivo my-project-nameUITest-runner.app my-project-nameUITestou.ipa e os arquivos de código-fonte do xcode necessários para executar o XCTEST_UI com planos de teste, incluindo um arquivo ou. `.xcworkspace` `.xcodeproj`

Aqui está um exemplo de zip usando um `.xcodeproj` arquivo:

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

Aqui está um exemplo de zip usando um `.xcworkspace` arquivo:

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

> **Note**
>
> Certifique-se de que você não tenha um diretório chamado “Payload” dentro do seu pacote XCTest UI .zip.

Carregue seus testes de XCTest interface do iOS

Use o console do Device Farm para carregar seus testes.

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e, em seguida, Projetos.
3. Na lista de projetos, escolha o projeto para o qual deseja carregar seus testes.

   > **Tip**
   >
   > Você pode usar a barra de pesquisa para filtrar a lista de projetos por nome.
   > Para criar um projeto, siga as instruções em Criação de um projeto no AWS Device Farm.

4. Escolha Criar execução.
5. Em Configurações de execução, na seção Tipo de execução, escolha aplicativo para iOS.
6. Em Selecionar aplicativo, na seção Opções de seleção de aplicativos, selecione Carregar seu próprio aplicativo. Em seguida, selecione Escolher arquivo em Carregar aplicativo.
7. Procure e escolha o arquivo de seu aplicativo iOS. O arquivo deve ser .ipa.

> **Note**
>
> Confirme se o arquivo .ipa foi desenvolvido para um dispositivo iOS e não para um simulador.

8. Em Configurar teste, na seção Selecionar estrutura de teste, escolha XCTest UI. Em seguida, selecione Escolher arquivo em Carregar aplicativo.
9. Navegue até o arquivo.ipa ou.zip que contém o executor de testes da XCTest interface do usuário do iOS e o escolha.
10. Conclua as etapas restantes no processo de criação da execução. Você selecionará os dispositivos nos quais deseja testar e, opcionalmente, especificará uma configuração adicional.
11. Escolha Criar execução. O Device Farm executa o teste e mostra os resultados no console.

Testes de aplicativos web no AWS Device Farm

O Device Farm fornece testes com o Appium para aplicações Web. Para obter mais informações sobre como configurar testes do Appium no Device Farm, consulte the section called “Appium”.

Para obter mais informações sobre testes no Device Farm, consulteEstruturas de teste e testes integrados no AWS Device Farm.

Regras para dispositivos de acesso limitado e ilimitado

A medição refere-se à cobrança dos dispositivos. Por padrão, os dispositivos do Device Farm são medidos e você é cobrado por minuto depois que os minutos da avaliação gratuita são usados. Você também pode optar por comprar dispositivos de acesso ilimitado, que permite um número ilimitado de testes mediante o pagamento de uma taxa mensal fixa. Para obter mais informações sobre preços, consulte Definição de preço do AWS Device Farm.

Se você optar por iniciar uma execução com um grupo de dispositivos que contém dispositivos iOS e Android, há regras para dispositivos de acesso limitado e ilimitado. Por exemplo, se você tiver cinco dispositivos Android de acesso ilimitado e cinco dispositivos iOS de acesso ilimitado, as execuções de testes web usarão os dispositivos de acesso ilimitado.

Veja outro exemplo: suponhamos que você tenha cinco dispositivos Android de acesso ilimitado e 0 dispositivo iOS de acesso ilimitado. Se selecionar apenas dispositivos Android para a execução web,

seus dispositivos de acesso ilimitado serão usados. Se selecionar dispositivos Android e iOS para a execução web, o método de cobrança será monitorado e seus dispositivos de acesso ilimitado não serão usados.

Testes integrados no AWS Device Farm

O Device Farm oferece suporte a tipos de teste incorporados para dispositivos Android e iOS.

Com os testes integrados, você pode testar a aplicação em vários dispositivos sem precisar escrever e manter scripts de automação de testes. Isso pode economizar tempo e esforço, especialmente quando você está começando a usar o Device Farm. O Device Farm oferece o seguinte tipo de teste integrado:

- Integrado: Fuzz (Android e iOS)— O teste de fuzz envia aleatoriamente eventos da interface do usuário para os dispositivos e, em seguida, relata os resultados.

Para obter mais informações sobre testes e estruturas de teste no Device Farm, consulteEstruturas de teste e testes integrados no AWS Device Farm.

Executando o teste de fuzz integrado do Device Farm (Android e iOS)

O teste de difusão integrado do Device Farm envia aleatoriamente eventos da interface do usuário para os dispositivos e, em seguida, relata os resultados.

Para obter mais informações sobre testes no Device Farm, consulteEstruturas de teste e testes integrados no AWS Device Farm.

Para executar o teste de fuzz integrado

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e, em seguida, Projetos.
3. Na lista de projetos, escolha o projeto em que você deseja executar o teste de fuzz incorporado.

> **Tip**
>
> Você pode usar a barra de pesquisa para filtrar a lista de projetos por nome.

Para criar um projeto, siga as instruções em Criação de um projeto no AWS Device Farm.

4. Escolha Criar execução.
5. Em Configurações de execução, selecione seu tipo de execução na seção Tipo de execução. Selecione aplicativo Android se você não tiver um aplicativo pronto para teste ou se estiver testando um aplicativo Android (.apk). Selecione o aplicativo iOS se estiver testando um aplicativo iOS (.ipa).
6. Em Selecionar aplicativo, escolha Selecionar aplicativo de amostra fornecido pelo Device Farm se você não tiver um aplicativo disponível para teste. Se você estiver trazendo seu próprio aplicativo, selecione Carregar próprio aplicativo e escolha o arquivo do seu aplicativo.
7. Em Configurar teste, na seção Selecionar estrutura de teste, escolha Integrado: Fuzz.
8. Se qualquer uma das configurações a seguir forem exibidas, você poderá aceitar os valores padrão ou especificar seu próprio:

   - Contagem de eventos: especifique um número entre 1 e 10.000, que representa o número de eventos de interface de usuário que o teste Fuzz deve executar.
   - Restrição de eventos: especifique um número entre 0 e 1.000, que representa o número de milissegundos que o teste de fuzz deve aguardar para realizar o próximo evento de interface de usuário.
   - Propagação aleatória: especifique um número para o teste de fuzz usar para randomizar eventos de interface de usuário. A especificação de um mesmo número de testes Fuzz subsequentes garante sequências de eventos idênticas.
9. Complete as instruções restantes para selecionar dispositivos e iniciar a execução.

Ambientes de teste personalizados no AWS Device Farm

O AWS Device Farm permite configurar um ambiente personalizado para testes automatizados (modo personalizado), que é a abordagem recomendada para todos os usuários do Device Farm. Para saber mais sobre ambientes no Device Farm, consulte Test environments.

Os benefícios do Modo Personalizado em oposição ao Modo Padrão incluem:

- Execução mais rápida do end-to-end teste: o pacote de teste não é analisado para detectar todos os testes na suíte, evitando a sobrecarga de pré-processamento/pós-processamento.
- Log ao vivo e streaming de vídeo: os log de teste e vídeo do lado do cliente são transmitidos ao vivo ao usar o modo personalizado. Esse recurso não está disponível na modalidade padrão.
- Captura todos os artefatos: no host e no dispositivo, o modo personalizado permite capturar todos os artefatos de teste. Isso pode não ser possível no modo padrão.
- Ambiente local mais consistente e replicável: no modo padrão, os artefatos serão fornecidos para cada teste individual separadamente, o que pode ser benéfico em determinadas circunstâncias. No entanto, seu ambiente de teste local pode se desviar da configuração original, pois o Device Farm trata cada teste executado de forma diferente.

  Por outro lado, o Modo Personalizado permite que você torne seu ambiente de execução de testes do Device Farm consistentemente alinhado com seu ambiente de teste local.

Ambientes personalizados são configurados usando um arquivo de especificação de teste formatado em YAML (especificação de teste). O Device Farm fornece um arquivo de especificação de teste padrão para cada tipo de teste compatível que pode ser usado como está ou personalizado; personalizações como filtros de teste ou arquivos de configuração podem ser adicionadas à especificação de teste. As especificações de teste editadas podem ser salvas para futuros testes.

Para obter mais informações, consulte Uploading a Custom Test Spec Using the AWS CLI e Criando uma execução de teste no Device Farm.

Tópicos

- Teste a sintaxe da especificação no Device Farm
- Exemplo do arquivo de especificação de teste do Device Farm
- Ambiente de teste Amazon Linux 2 para testes de Android
- Variáveis de ambiente no Device Farm

- Migração de testes de um ambiente de teste padrão para um ambiente de teste personalizado
- Extensão de ambientes de teste personalizados no Device Farm

Teste a sintaxe da especificação no Device Farm

A especificação de teste é um arquivo que você usa para definir ambientes de teste personalizados no AWS Device Farm. Para obter mais informações sobre os ambientes personalizados e o arquivo de especificações de teste, consulteAmbientes de teste personalizados no AWS Device Farm.

A seguir está a estrutura do arquivo de especificação de teste YAML. Seguindo a estrutura, há uma descrição de cada propriedade.

Para ver um exemplo de arquivo de especificação de teste, consulte. Exemplo do arquivo de especificação de teste do Device Farm

```
version: 0.1

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

artifacts:
  - location
  - location
```

A especificação de teste contém o seguinte:

`version`

Reflete a versão da especificação de teste compatível com o Device Farm. O número da versão atual é 0.1.

`phases`

Esta seção contém grupos de comandos executados durante uma execução de teste.

Os nomes da fase de teste permitidos são:

`install`

Opcional.

As dependências padrão para estruturas de teste compatíveis com o Device Farm já estão instaladas. Essa fase contém comandos adicionais, se houver, que o Device Farm executa durante a instalação.

`pre_test`

Opcional.

Os comandos, se houver, executados antes da execução de teste automatizada.

`test`

Opcional.

Os comandos executados durante a execução de teste automatizada. Se qualquer comando na fase de teste falhar, o teste será marcado como falha.

`post_test`

Opcional.

Os comandos, se houver, executados depois da execução de teste automatizada.

`artifacts`

Opcional.

O Device Farm reúne artefatos como relatórios personalizados, arquivos de log e imagens de um local especificado aqui. Os caracteres curinga não são permitidos como parte de um artefato local. Dessa forma, você deve especificar um caminho válido para cada local.

Esses artefatos de teste estão disponíveis para cada dispositivo na execução de teste. Para obter informações sobre como recuperar os artefatos de teste, consulte Baixando artefatos em um ambiente de teste personalizado.

> **Important**
>
> Uma especificação de teste deve ser formatada como um arquivo YAML válido. Caso o recuo ou o espaçamento na especificação de teste seja inválido, a execução de teste pode falhar. As guias não são permitidas em arquivos YAML. Você pode usar um validador YAML para testar se a especificação de teste é um arquivo YAML válido. Para obter mais informações, consulte o site da YAML.

Exemplo do arquivo de especificação de teste do Device Farm

A especificação de teste é um arquivo que você usa para definir ambientes de teste personalizados no AWS Device Farm. Para obter mais informações sobre os ambientes personalizados e o arquivo de especificações de teste, consulte.. Ambientes de teste personalizados no AWS Device Farm Para obter mais informações sobre a estrutura e o conteúdo do arquivo de especificação de teste, consulteTeste a sintaxe da especificação no Device Farm.

Veja a seguir um exemplo de uma especificação de teste YAML do Device Farm que configura uma execução de teste Appium Java TestNG.

```
version: 0.1

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

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

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

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

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

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

```
        devicefarm-cli use appium 2;
      else
        # For iOS, Appium versions 1.22.2 and 2.2.1 are preinstalled and selectable
through avm.
        # For all other versions, please use npm to install them. For example:
        # npm install -g appium@2.1.3;
        # Note that, for iOS devices, Appium 2 is only supported on iOS version 14
and above using
        # NodeJS version 16 and above.
        avm 2.2.1;
      fi;
    - appium --version

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

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

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

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

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

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

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

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

```
            \"appium:automationName\": \"XCUITest\"}" \
            >> $DEVICEFARM_LOG_DIR/appium.log 2>&1 &
        fi;

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

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

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

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

# Artifacts are a list of paths on the filesystem where you can store test output and
 reports.
# All files in these paths will be collected by Device Farm.
```

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

Ambiente de teste Amazon Linux 2 para testes de Android

O AWS Device Farm utiliza máquinas host Amazon Elastic Compute Cloud (EC2) executando o Amazon Linux 2 para executar testes do Android. Quando você agenda uma execução de teste, o Device Farm aloca um host dedicado para cada dispositivo para executar testes de forma independente. As máquinas host são encerradas após a execução do teste junto com todos os artefatos gerados.

O host de teste do Amazon Linux 2 é o mais novo ambiente de teste do Android, substituindo o sistema anterior baseado no Ubuntu. Usando seu arquivo de especificação de teste, você pode optar por executar seus testes do Android no ambiente Amazon Linux 2.

O host Amazon Linux 2 oferece várias vantagens:

- Testes mais rápidos e confiáveis: em comparação com o host antigo, o novo host de teste melhora significativamente a velocidade do teste, especialmente reduzindo os tempos de início do teste. O host Amazon Linux 2 também demonstra maior estabilidade e confiabilidade durante os testes.
- Acesso remoto aprimorado para testes manuais: as atualizações para o host de teste mais recente e as melhorias resultam em menor latência e melhor desempenho de vídeo para testes manuais do Android.
- Seleção de versão de software padrão: o Device Farm agora padroniza o suporte às principais linguagens de programação no host de teste, bem como nas versões de estrutura do Appium. Para linguagens compatíveis (atualmente Java, Python, Node.js e Ruby) e Appium, o novo host de teste fornece versões estáveis de longo prazo logo após o lançamento. O gerenciamento centralizado de versões por meio da ferramenta `devicefarm-cli` permite o desenvolvimento de arquivos de especificações de teste com uma experiência consistente entre as estruturas.

Tópicos

- Bibliotecas de software pré-instaladas para dar suporte aos testes Device Farm de dispositivos Android

- Intervalos de IP suportados para o ambiente de teste Amazon Linux 2 no Device Farm
- Usando a devicefarm-cli ferramenta no AWS Device Farm
- Selecionar qual host de teste do Android usar no Device Farm
- Exemplo: arquivo de especificação de teste do Device Farm mostrando um host de teste do Android
- Migração para o host de teste do Amazon Linux 2 no AWS Device Farm

Bibliotecas de software pré-instaladas para dar suporte aos testes Device Farm de dispositivos Android

O AWS Device Farm usa máquinas host Amazon Elastic Compute Cloud (EC2) executando o Amazon Linux 2 para executar testes do Android. O host de teste do Amazon Linux 2 vem pré-instalado com muitas das bibliotecas de software necessárias para permitir as estruturas de teste do Device Farm, fornecendo um ambiente de teste pronto no lançamento. Para qualquer outro software necessário, você pode modificar o arquivo de especificação de teste para instalar pelo pacote de testes, fazer o download da Internet ou acessar origens privadas na VPC (consulte VPC ENI para obter mais informações). Para obter mais informações, consulte Test spec file example.

As seguintes versões de software estão atualmente disponíveis no host:

Biblioteca de software	Versão do software	Comando a ser usado em seu arquivo de especificação de teste
Python	3.8	`devicefarm-cli use python 3.8`
	3.9	`devicefarm-cli use python 3.9`
	3.10	`devicefarm-cli use python 3.10`
	3.11	`devicefarm-cli use python 3.11`

Java	8	`devicefarm-cli use java 8`
	11	`devicefarm-cli use java 11`
	17	`devicefarm-cli use java 17`
NodeJS	16	`devicefarm-cli use node 16`
	18	`devicefarm-cli use node 18`
	20	`devicefarm-cli use node 20`
Ruby	2.7	`devicefarm-cli use ruby 2.7`
	3.2	`devicefarm-cli use ruby 3.2`
Appium	1	`devicefarm-cli use appium 1`
	2	`devicefarm-cli use appium 2`

O host de teste também inclui ferramentas de suporte comumente usadas para cada versão do software, como gerenciadores de `npm` pacotes `pip` e (incluídos no Python e no Node.js, respectivamente) e dependências (como o UIAutomator2 driver Appium) para ferramentas como o Appium. Isso garante que você tenha as ferramentas necessárias para trabalhar com as estruturas de teste compatíveis.

Intervalos de IP suportados para o ambiente de teste Amazon Linux 2 no Device Farm

Os clientes geralmente precisam saber o intervalo de IP do qual o tráfego do Device Farm se origina, principalmente para definir seus firewalls e configurações de segurança. Para os hosts EC2 de teste da Amazon, o intervalo de IP abrange toda `us-west-2` a região. Para os hosts de teste do Amazon Linux 2, que é a opção padrão para novas execuções do Android, os intervalos foram restritos. O tráfego agora se origina de um conjunto específico de gateways NAT, restringindo o intervalo de IP aos seguintes endereços:

Intervalos de IP
44.236.137.143
52.13.151.244
52.35.189.191
54.201.250.26

Para obter mais informações sobre ambientes de teste do Android no Device Farm, consulteAmbiente de teste Amazon Linux 2 para testes de Android.

Usando a **`devicefarm-cli`** ferramenta no AWS Device Farm

O AWS Device Farm usa máquinas host Amazon Elastic Compute Cloud (EC2) executando o Amazon Linux 2 para executar testes do Android. O host de teste do Amazon Linux 2 usa uma ferramenta de gerenciamento de versão padronizada chamada `devicefarm-cli` para selecionar versões de software. Essa ferramenta é separada da AWS CLI e está disponível somente no Device Farm Test Host. Com a `devicefarm-cli`, você pode alternar para qualquer versão de software pré-instalada no host de teste. Isso proporciona uma maneira direta de manter o arquivo de especificações de teste do Device Farm ao longo do tempo e oferece um mecanismo previsível para atualizar as versões do software no futuro.

O trecho abaixo mostra a página de `help` da `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.
```

Vamos analisar alguns exemplos usando a `devicefarm-cli`. Para usar a ferramenta para alterar a versão do Python de *`3.10`* para *`3.9`* em seu arquivo de especificação de teste, execute os seguintes comandos:

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

Para alterar a versão do Appium de *`1`* para: *`2`*

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

> **Tip**
>
> Observe que, quando você seleciona uma versão de software, a `devicefarm-cli` também alterna as ferramentas de suporte para essas linguagens, como `pip` para Python e `npm` para NodeJS.

Para obter mais informações sobre como o Device Farm testa dispositivos Android, consulteAmbiente de teste Amazon Linux 2 para testes de Android.

Para obter mais informações sobre o software pré-instalado no host de teste do Amazon Linux 2, consulteBibliotecas de software pré-instaladas para dar suporte aos testes Device Farm de dispositivos Android.

Selecionar qual host de teste do Android usar no Device Farm

> **Warning**
>
> O antigo Android Test Host não estará mais disponível em 21 de outubro de 2024. Observe que o processo de suspensão de uso está dividido em várias datas:
>
> - Em 22 de abril de 2024, as tarefas de qualquer nova conta serão direcionadas para o host de teste atualizado.
> - Em 2 de setembro de 2024, todos os arquivos de especificações de teste novos ou modificados devem ter como alvo o host de teste atualizado.
> - Em 21 de outubro de 2024, os trabalhos não poderão mais ser executados no host de teste legado.
>
> Configure seus arquivos de especificações de teste para o `amazon_linux_2` host para evitar problemas de compatibilidade.
> Observe que o Legacy Android Test Host suporta apenas as versões 14 e inferiores do Android. Use o host amazon_linux_2 para as versões 15 e superiores do Android.

O AWS Device Farm usa máquinas host Amazon Elastic Compute Cloud (EC2) executando o Amazon Linux 2 para executar testes do Android. Para testes de Android, o Device Farm requer o seguinte campo em seu arquivo de especificações de teste para escolher o host de teste do Amazon Linux 2:

```
android_test_host: amazon_linux_2 | legacy
```

Use o `amazon_linux_2` para executar seus testes no host de teste do Amazon Linux 2:

```
android_test_host: amazon_linux_2
```

Saiba mais sobre os benefícios do Amazon Linux 2 aqui.

O Device Farm recomenda o uso do host Amazon Linux 2 para testes de Android em vez do ambiente de host legado. Se você preferir usar o ambiente legado, use `legacy` para executar seus testes no host de teste legado:

```
android_test_host: legacy
```

Por padrão, os arquivos de especificação de teste sem uma seleção de host de teste serão executados no host de teste legado.

Sintaxe descontinuada

Abaixo está a sintaxe obsoleta para escolher o Amazon Linux 2 em seu arquivo de especificação de teste:

```
preview_features:
  android_amazon_linux_2_host: true
```

Se você estiver usando esse sinalizador, seus testes continuarão a ser executados no Amazon Linux 2. No entanto, é altamente recomendável remover a seção de sinalizadores `preview_features` e substituí-la pelo novo campo `android_test_host` para evitar a sobrecarga de manutenção no futuro.

> **Warning**
>
> Usar os sinalizadores `android_test_host` e `android_amazon_linux_2_host` em seu arquivo de especificação de teste retornará um erro. Somente um deve ser usado. Recomendamos `android_test_host`.

Exemplo: arquivo de especificação de teste do Device Farm mostrando um host de teste do Android

O trecho a seguir é um exemplo de um arquivo de especificação de teste do Device Farm que configura uma execução de teste do Appium NodeJS usando o host de teste do Amazon Linux 2 para 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>&amp;1 &amp;

      # 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
```

Migração para o host de teste do Amazon Linux 2 no AWS Device Farm

> **Warning**
>
> O antigo Android Test Host não estará mais disponível em 21 de outubro de 2024. Observe que o processo de suspensão de uso está dividido em várias datas:
>
> - Em 22 de abril de 2024, as tarefas de qualquer nova conta serão direcionadas para o host de teste atualizado.
> - Em 2 de setembro de 2024, todos os arquivos de especificações de teste novos ou modificados devem ter como alvo o host de teste atualizado.
> - Em 21 de outubro de 2024, os trabalhos não poderão mais ser executados no host de teste legado.
>
> Configure seus arquivos de especificações de teste para o `amazon_linux_2` host para evitar problemas de compatibilidade.

Para migrar os testes existentes do host legado para o novo host Amazon Linux 2, desenvolva novos arquivos de especificação de teste com base nos arquivos pré-existentes. A abordagem

recomendada é começar com os novos arquivos de especificações de teste padrão para seus tipos de teste. Em seguida, migre os comandos relevantes do arquivo de especificações de teste antigo para o novo, salvando o arquivo antigo como backup. Isso permite que você aproveite a especificação padrão otimizada para o novo host enquanto reutiliza seu código existente. Isso garante que você obtenha todos os benefícios do novo host configurado de forma otimizada para os seus testes e, ao mesmo tempo, mantenha a especificação de teste herdada como referência enquanto adapta os comandos ao novo ambiente.

As etapas a seguir podem ser usadas para criar um novo arquivo de especificações de teste do Amazon Linux 2 e, ao mesmo tempo, reutilizar os comandos do arquivo de especificações de teste antigo:

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. Navegue até o projeto do Device Farm que contém seus testes de automação.
3. Escolha Criar execução no projeto.
4. Escolha uma aplicação e um pacote de teste usados anteriormente para sua estrutura de teste.
5. Escolha Executar teste em um ambiente personalizado.
6. Escolha o arquivo de especificações de teste que você está usando atualmente para testes no host de teste legado no menu suspenso de especificações de teste.
7. Copie o conteúdo desse arquivo e cole-o localmente em um editor de texto para referência posterior.
8. No menu suspenso da especificação de teste, altere sua seleção de especificação de teste para o arquivo de especificação de teste padrão mais recente.
9. Escolha Editar e você entrará na interface de edição da especificação de teste. Você perceberá que, nas primeiras linhas do arquivo de especificações de teste, ele já optou pelo novo host de teste:

   ```
   android_test_host: amazon_linux_2
   ```

10. Revise a sintaxe para selecionar hosts de teste aqui e as principais diferenças entre os hosts de teste aqui.
11. Adicione e edite seletivamente os comandos do seu arquivo de especificações de teste salvo localmente na etapa 6 no novo arquivo de especificações de teste padrão. Em seguida, escolha Salvar como para salvar o novo arquivo de especificação. Agora você pode agendar execuções de teste no host de teste do Amazon Linux 2.

Diferenças entre os hosts de teste novos e os antigos

Ao editar o arquivo de especificações de teste para usar o host de teste do Amazon Linux 2 e fazer a transição dos testes do host de teste legado, esteja ciente dessas diferenças importantes de ambiente:

- Seleção de versões de software: em muitos casos, as versões padrão do software foram alteradas. Portanto, se você não estava selecionando explicitamente sua versão de software no host de teste legado antes, talvez queira especificá-la agora no host de teste do Amazon Linux 2 usando `devicefarm-cli`. Na grande maioria dos casos de uso, recomendamos que os clientes selecionem explicitamente as versões do software que usam. Ao selecionar uma versão de software com `devicefarm-cli`, você terá uma experiência previsível e consistente com ela e receberá muitos avisos se o Device Farm planejar remover essa versão do host de teste.

  Além disso, ferramentas de seleção de software como `nvm`, `pyenv`, `avm` e `rvm` foram removidas em favor do novo sistema de seleção de software `devicefarm-cli`.
- Versões de software disponíveis: muitas versões do software pré-instalado anteriormente foram removidas e muitas novas versões foram adicionadas. Portanto, ao usar a `devicefarm-cli` para selecionar suas versões de software, você deve selecionar as versões que estão na lista de versões compatíveis.
- Qualquer caminho de arquivo codificado em seu arquivo de especificação de teste do host Legacy como caminhos absolutos provavelmente não funcionará conforme o esperado no host de teste do Amazon Linux 2; geralmente, eles não são recomendados para o uso de arquivos de especificações de teste. Recomendamos que você use caminhos relativos e variáveis de ambiente para todo o código do arquivo de especificação de teste. Além disso, observe que a maioria dos binários necessários para o teste podem ser encontrados no PATH do host para que possam ser executados imediatamente a partir do arquivo de especificação usando apenas o nome (como appium).
- No momento, não há suporte para a coleta de dados de desempenho no novo host de teste.
- Versão do sistema operacional: o host de teste legado foi baseado no sistema operacional Ubuntu, enquanto o novo é baseado no Amazon Linux 2. Como resultado, os usuários podem notar algumas diferenças nas bibliotecas do sistema e nas versões da biblioteca do sistema disponíveis.
- Para usuários do Appium Java, o novo host de teste não contém nenhum arquivo JAR pré-instalado em seu caminho de classe, enquanto o host anterior continha um para a estrutura TestNG (por meio de uma variável de ambiente`$DEVICEFARM_TESTNG_JAR`). Recomendamos que os clientes empacotem os arquivos JAR necessários para suas estruturas de teste dentro do

pacote de teste e removam instâncias da variável `$DEVICEFARM_TESTNG_JAR` de seus arquivos de especificação de teste. Para obter mais informações, consulte Trabalhar com o Appium e o AWS Device Farm.

- Para usuários do Appium, a variável de ambiente `$DEVICEFARM_CHROMEDRIVER_EXECUTABLE` foi removida em favor de uma nova abordagem para permitir que os clientes acessem o Chromedriver para Android. Veja nosso arquivo de especificação de teste padrão para ver um exemplo, que usa uma nova variável de ambiente `$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR`.

> **Note**
>
> É altamente recomendável manter o comando do servidor Appium existente no arquivo de especificação de teste padrão como está.

Recomendamos que entre em contato com a equipe de serviço por meio de um caso de suporte se tiver algum feedback ou dúvida sobre as diferenças entre os hosts de teste do ponto de vista do software.

Variáveis de ambiente no Device Farm

As variáveis de ambiente representam valores usados pelos testes automatizados. Você pode usar essas variáveis de ambiente nos arquivos YAML e no código de teste. Em um ambiente de teste personalizado, o Device Farm preenche dinamicamente as variáveis de ambiente no runtime.

Tópicos

- Variáveis de ambiente comuns no Device Farm
- Variáveis de JUnit ambiente Appium Java no Device Farm
- Variáveis de ambiente Appium Java TestNG em Device Farm
- XCUITest variáveis de ambiente no Device Farm

Variáveis de ambiente comuns no Device Farm

Esta seção descreve as variáveis de ambiente que são comuns aos testes da plataforma Android e aos testes da plataforma iOS no AWS Device Farm. Para obter mais informações sobre variáveis de ambiente no Device Farm, consulteVariáveis de ambiente no Device Farm.

Testes do Android

Esta seção descreve as variáveis de ambiente personalizadas comuns aos testes da plataforma Android compatíveis com o Device Farm.

`$DEVICEFARM_DEVICE_NAME`

Nome do dispositivo no qual os testes são executados. Ele representa o Unique Device Identifer (UDID – Identificador exclusivo do dispositivo).

`$DEVICEFARM_DEVICE_PLATFORM_NAME`

O nome da plataforma do dispositivo. Ele é Android ou iOS.

`$DEVICEFARM_DEVICE_OS_VERSION`

A versão do sistema operacional do dispositivo.

`$DEVICEFARM_APP_PATH`

O caminho do aplicativo para dispositivos móveis na máquina de host onde os testes estão sendo executados. O caminho do aplicativo só está disponível para aplicativos para dispositivos móveis.

`$DEVICEFARM_DEVICE_UDID`

O identificador exclusivo do dispositivo para aplicativos móveis que executam o teste automatizado.

`$DEVICEFARM_LOG_DIR`

O caminho para os arquivos de log gerados durante a execução de teste. Por padrão, todos os arquivos nesse diretório são arquivados em um arquivo ZIP e disponibilizados como artefato após a execução do teste.

`$DEVICEFARM_SCREENSHOT_PATH`

O caminho das capturas de telas, se houver, capturadas durante a execução de teste.

`$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR`

A localização de um diretório que contém os executáveis necessários do Chromedriver para uso nos testes web e híbridos do Appium.

`$ANDROID_HOME`

O caminho para o diretório de instalação do SDK do Android.

> **Note**
>
> A variável de ambiente `ANDROID_HOME` só está disponível no host de teste do Amazon Linux 2 para Android.

Testes do iOS

Esta seção descreve as variáveis de ambiente personalizadas comuns aos testes da plataforma iOS compatíveis com o Device Farm.

`$DEVICEFARM_DEVICE_NAME`

Nome do dispositivo no qual os testes são executados. Ele representa o Unique Device Identifer (UDID – Identificador exclusivo do dispositivo).

`$DEVICEFARM_DEVICE_PLATFORM_NAME`

O nome da plataforma do dispositivo. Ele é Android ou iOS.

`$DEVICEFARM_APP_PATH`

O caminho do aplicativo para dispositivos móveis na máquina de host onde os testes estão sendo executados. O caminho do aplicativo só está disponível para aplicativos para dispositivos móveis.

`$DEVICEFARM_DEVICE_UDID`

O identificador exclusivo do dispositivo para aplicativos móveis que executam o teste automatizado.

`$DEVICEFARM_LOG_DIR`

O caminho para os arquivos de log gerados durante a execução de teste.

`$DEVICEFARM_SCREENSHOT_PATH`

O caminho das capturas de telas, se houver, capturadas durante a execução de teste.

Variáveis de JUnit ambiente Appium Java no Device Farm

Esta seção descreve as variáveis de ambiente usadas pelos JUnit testes Appium Java em um ambiente de teste personalizado no AWS Device Farm. Para obter mais informações sobre variáveis de ambiente no Device Farm, consulteVariáveis de ambiente no Device Farm.

$DEVICEFARM_TESTNG_JAR

O caminho do arquivo .jar do TestNG.

$DEVICEFARM_TEST_PACKAGE_PATH

O caminho para o conteúdo descompactado do arquivo do pacote de teste.

Variáveis de ambiente Appium Java TestNG em Device Farm

Esta seção descreve as variáveis de ambiente usadas pelos testes Appium Java TestNG em um ambiente de teste personalizado no Device Farm. Para obter mais informações sobre variáveis de ambiente no Device Farm, consulteVariáveis de ambiente no Device Farm.

$DEVICEFARM_TESTNG_JAR

O caminho do arquivo .jar do TestNG.

$DEVICEFARM_TEST_PACKAGE_PATH

O caminho para o conteúdo descompactado do arquivo do pacote de teste.

XCUITest variáveis de ambiente no Device Farm

Esta seção descreve as variáveis de ambiente usadas pelo XCUITest teste em um ambiente de teste personalizado no Device Farm. Para obter mais informações sobre variáveis de ambiente no Device Farm, consulteVariáveis de ambiente no Device Farm.

$DEVICEFARM_XCUITESTRUN_FILE

Caminho para o arquivo `.xctestun` do Device Farm. Ele é gerado com base nos pacotes de aplicativos e testes.

$DEVICEFARM_DERIVED_DATA_PATH

Caminho esperado da saída do xcodebuild do Device Farm.

$DEVICEFARM_XCTEST_BUILD_DIRECTORY

O caminho para o conteúdo descompactado do arquivo do pacote de teste.

Migração de testes de um ambiente de teste padrão para um ambiente de teste personalizado

Você pode alternar de um modo de execução de teste padrão para um modo de execução personalizado no AWS Device Farm. A migração envolve principalmente duas formas diferentes de execução:

1. Modo padrão: esse modo de execução de teste foi criado principalmente para fornecer aos clientes relatórios granulares e um ambiente totalmente gerenciado.
2. Modo personalizado: esse modo de execução de teste foi criado para diferentes casos de uso que exigem execuções de teste mais rápidas, a capacidade de mover sem alterações (lift-and-shift) e alcançar a paridade com o ambiente local e streaming de vídeo ao vivo.

Para obter mais informações sobre os modos padrão e personalizado no Device Farm, consulte Ambientes de teste no AWS Device Farm Ambientes de teste personalizados no AWS Device Farm e.

Considerações ao migrar

Esta seção lista alguns dos principais casos de uso a serem considerados ao migrar para o modo personalizado:

1. Velocidade: no modo padrão de execução, o Device Farm analisa os metadados dos testes que você empacotou e carregou usando as instruções de empacotamento de sua estrutura específica. A análise detecta o número de testes em seu pacote. Depois disso, o Device Farm executa cada teste separadamente e apresenta os logs, vídeos e outros artefatos de resultados individualmente para cada teste. No entanto, isso aumenta constantemente o tempo total de execução do end-to-end teste, pois há o pré e o pós-processamento de testes e artefatos de resultados no final do serviço.

   Por outro lado, o modo de execução personalizado não analisa seu pacote de teste; isso significa nenhum pré-processamento e um pós-processamento mínimo para testes ou artefatos de resultados. Isso resulta em tempos totais de end-to-end execução próximos à sua configuração local. Os testes são executados no mesmo formato em que seriam se fossem executados nas máquinas locais. Os resultados dos testes são os mesmos que você obtém localmente e estão disponíveis para download no final da execução do trabalho.

2. Personalização ou flexibilidade: o modo padrão de execução analisa seu pacote de teste para detectar o número de testes e, em seguida, executa cada teste separadamente. Observe que não há garantia de que os testes serão executados na ordem especificada. Como resultado, os testes que exigem uma sequência específica de execução podem não funcionar conforme o esperado. Além disso, não há como personalizar o ambiente da máquina host ou passar arquivos de configuração que possam ser necessários para executar seus testes de uma determinada maneira.

   Por outro lado, o modo personalizado permite que você configure o ambiente da máquina host, incluindo a capacidade de instalar software adicional, passar filtros para seus testes, passar arquivos de configuração e controlar a configuração da execução do teste. Ele consegue isso por meio de um arquivo yaml (também chamado de arquivo testspec) que você pode modificar adicionando comandos shell a ele. Esse arquivo yaml é convertido em um script de shell que é executado na máquina host de teste. Você pode salvar vários arquivos yaml e escolher um dinamicamente de acordo com seus requisitos ao agendar uma execução.
3. Vídeo ao vivo e registro em log: os modos de execução padrão e personalizado fornecem vídeos e logs para seus testes. No entanto, no modo padrão, você obtém o vídeo e os logs predefinidos de seus testes somente após a conclusão dos testes.

   Por outro lado, o modo personalizado oferece uma transmissão ao vivo do vídeo e dos logs de seus testes do lado do cliente. Além disso, você pode baixar o vídeo e outros artefatos ao final dos testes.

> **Tip**
>
> Se seu caso de uso envolver pelo menos um dos fatores acima, é altamente recomendável mudar para o modo de execução personalizado.

Etapas da migração

Para migrar do modo Padrão para o modo Personalizado, faça o seguinte:

1. Faça login AWS Management Console e abra o console do Device Farm em https://console.aws.amazon.com/devicefarm/.
2. Escolha seu projeto e, em seguida, inicie uma nova execução de automação.

3. Faça upload da aplicação (ou selecione `web app`), escolha o tipo de estrutura de teste, faça upload do pacote de teste e, no parâmetro `Choose your execution environment`, escolha a opção para `Run your test in a custom environment`.
4. Por padrão, o arquivo de exemplo de especificação de teste do Device Farm aparecerá para você visualizar e editar. Esse arquivo de exemplo pode ser usado como ponto de partida para testar seus testes no modo de ambiente personalizado. Depois de verificar se seus testes estão funcionando corretamente no console, você pode alterar qualquer uma de suas integrações de API, CLI e pipeline com o Device Farm para usar esse arquivo de especificação de teste como parâmetro ao programar execuções de teste. Para obter informações sobre como adicionar um arquivo de especificação de teste como parâmetro para suas execuções, consulte a seção do parâmetro `testSpecArn` da API `ScheduleRun` em nosso Guia de API.

Estrutura do Appium

Em um ambiente de teste personalizado, o Device Farm não insere nem substitui nenhum recurso do Appium em seus testes da estrutura do Appium. Você deve especificar os recursos do Appium do teste no arquivo YAML da especificação de teste ou no código de teste.

Instrumentação do Android

Você não precisa fazer alterações a fim de mover os testes de instrumentação do Android para um ambiente de teste personalizado.

iOS XCUITest

Você não precisa fazer alterações para mover seus XCUITest testes do iOS para um ambiente de teste personalizado.

Extensão de ambientes de teste personalizados no Device Farm

O AWS Device Farm permite configurar um ambiente personalizado para testes automatizados (modo personalizado), que é a abordagem recomendada para todos os usuários do Device Farm. O modo personalizado Device Farm permite que você execute mais do que apenas sua suíte de testes. Nesta seção, você aprenderá como estender sua suíte de testes e otimizar seus testes.

Para obter mais informações sobre ambientes de teste personalizados no Device Farm, consulteAmbientes de teste personalizados no AWS Device Farm.

Tópicos

- Configurando o PIN de um dispositivo ao executar testes no Device Farm
- Acelerando os testes baseados em Appium no Device Farm por meio dos recursos desejados
- Usando Webhooks e outros APIs após a execução dos testes no Device Farm
- Adicionar arquivos extras ao seu pacote de teste no Device Farm

Configurando o PIN de um dispositivo ao executar testes no Device Farm

Algumas aplicações exigem que você defina um PIN no dispositivo. O Device Farm não é compatível com a configuração nativa de um PIN em dispositivos. No entanto, isso é possível com as seguintes ressalvas:

- O dispositivo deve estar executando o Android 8 ou superior.
- O PIN deve ser removido após a conclusão do teste.

Para definir o PIN em seus testes, use as fases `pre_test` e `post_test` para definir e remover o PIN, conforme mostrado a seguir:

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

Quando o conjunto de testes é iniciado, o PIN 1234 é definido. Depois que sua suíte de testes sair, o PIN será removido.

> **Warning**
>
> Se você não remover o PIN do dispositivo após a conclusão do teste, o dispositivo e sua conta serão colocados em quarentena.

Para obter mais maneiras de ampliar sua suíte de testes e otimizar seus testes, consulteExtensão de ambientes de teste personalizados no Device Farm.

Acelerando os testes baseados em Appium no Device Farm por meio dos recursos desejados

Ao usar o Appium, você pode descobrir que o conjunto de testes do modo padrão é muito lento. Isso ocorre porque o Device Farm aplica as configurações padrão e não faz nenhuma suposição sobre como você deseja usar o ambiente Appium. Embora esses padrões sejam criados com base nas melhores práticas do setor, eles podem não se aplicar à sua situação. Para ajustar os parâmetros do servidor Appium, você pode ajustar os recursos padrão do Appium em sua especificação de teste. Por exemplo, o seguinte define o recurso `usePrebuildWDA` como `true` em um conjunto de testes do iOS para acelerar o tempo de início inicial:

```
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 &
```

Os recursos do Appium devem ser uma estrutura JSON citada e com escape de shell.

Os seguintes recursos do Appium são fontes comuns de melhorias de desempenho:

`noReset` e `fullReset`

Esses dois recursos, que são mutuamente exclusivos, descrevem o comportamento do Appium após a conclusão de cada sessão. Quando `noReset` está definido como `true`, o servidor Appium não remove dados da aplicação quando uma sessão do Appium termina, efetivamente não fazendo nenhuma limpeza. `fullReset` desinstala e limpa todos os dados da aplicação

do dispositivo após o encerramento da sessão. Para obter mais informações, consulte Reset Strategies na documentação do Appium.

`ignoreUnimportantViews` (somente Android)

Instrui o Appium a compactar a hierarquia da interface do usuário do Android somente em visualizações relevantes para o teste, acelerando as pesquisas de determinados elementos. No entanto, isso pode interromper alguns conjuntos de testes XPath baseados porque a hierarquia do layout da interface do usuário foi alterada.

`skipUnlock` (somente Android)

Informa à Appium que não há um código PIN definido atualmente, o que acelera os testes após um evento de desligamento da tela ou outro evento de bloqueio.

`webDriverAgentUrl` (somente iOS)

Instrui o Appium a assumir que uma dependência essencial do iOS, `webDriverAgent`, já está em execução e disponível para aceitar solicitações HTTP no URL especificado. Se `webDriverAgent` ainda não estiver instalado e funcionando, o Appium pode levar algum tempo no início de um conjunto de testes para iniciar o `webDriverAgent`. Se você iniciar `webDriverAgent` por conta própria e definir `webDriverAgentUrl` como `http://localhost:8100` ao iniciar o Appium, poderá inicializar o conjunto de testes mais rapidamente. Observe que esse recurso nunca deve ser usado junto com o recurso `useNewWDA`.

Você pode usar o código a seguir para iniciar `webDriverAgent` pelo arquivo de especificação de teste na porta local `8100` do dispositivo e, depois, encaminhá-lo para a porta local `8100` do host de teste (isso permite que você defina o valor de `webDriverAgentUrl` como `http://localhost:8100`). Esse código deve ser executado durante a fase de instalação após qualquer código para configurar o Appium e a definição das variáveis de ambiente `webDriverAgent`:

```
      # 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 &
```

Depois, você pode adicionar o código a seguir ao arquivo de especificação de teste para garantir que `webDriverAgent` tenha iniciado com êxito. Esse código deve ser executado no final da fase de pré-teste depois de garantir que o Appium tenha sido iniciado com sucesso:

```
      # 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;
```

Para obter mais informações sobre os recursos compatíveis com o Appium, consulte Appium Desired Capabilities na documentação do Appium.

Para obter mais maneiras de ampliar sua suíte de testes e otimizar seus testes, consulteExtensão de ambientes de teste personalizados no Device Farm.

Usando Webhooks e outros APIs após a execução dos testes no Device Farm

Você pode fazer com que o Device Farm chame um webhook depois que cada conjunto de testes terminar de usar curl. O processo para fazer isso varia de acordo com o destino e a formatação. Para

seu webhook específico, consulte a documentação desse webhook. O exemplo a seguir publica uma mensagem sempre que uma suíte de testes termina em um webhook do 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/XXXXXXXXXXXXXXXXXXXXXXX
```

Para obter mais informações sobre como usar webhooks com o Slack, consulte Sending your first Slack message using Webhook na referência de API do Slack.

Para obter mais maneiras de ampliar sua suíte de testes e otimizar seus testes, consulteExtensão de ambientes de teste personalizados no Device Farm.

Você não está limitado a usar curl para chamar webhooks. Os pacotes de teste podem incluir scripts e ferramentas extras, desde que sejam compatíveis com o ambiente de execução do Device Farm. Por exemplo, seu pacote de teste pode incluir scripts auxiliares que fazem solicitações a outros APIs. Certifique-se de que todos os pacotes necessários estejam instalados junto com os requisitos da sua suíte de testes. Para adicionar um script que seja executado após a conclusão da suíte de testes, inclua o script em seu pacote de teste e adicione o seguinte à sua especificação de teste:

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

> **Note**
>
> A manutenção de todas as chaves de API ou outros tokens de autenticação usados em seu pacote de teste é de sua responsabilidade. Recomendamos que você mantenha qualquer forma de credencial de segurança fora do controle de origem, use credenciais com o menor número possível de privilégios e use tokens revogáveis e de curta duração sempre que possível. Para verificar os requisitos de segurança, consulte a documentação do terceiro APIs que você usa.

Se você planeja usar AWS serviços como parte de sua suíte de execução de testes, você deve usar credenciais temporárias do IAM, geradas fora da suíte de testes e incluídas no pacote de teste. Essas credenciais devem ter o menor número de permissões concedidas e a menor vida

útil possível. Para obter mais informações sobre a criação de credenciais temporárias, consulte Requesting temporary security credentials no Guia do usuário do IAM.

Para obter mais maneiras de ampliar sua suíte de testes e otimizar seus testes, consulteExtensão de ambientes de teste personalizados no Device Farm.

Adicionar arquivos extras ao seu pacote de teste no Device Farm

Talvez você queira usar arquivos adicionais como parte de seus testes como arquivos extras de configuração ou dados de teste adicionais. Você pode incluir esses arquivos adicionais ao seu pacote de testes antes de carregá-lo no AWS Device Farm e acessá-los no modo de ambiente personalizado. Basicamente, todos os formatos de upload de pacotes de teste (ZIP, IPA, APK, JAR etc.) são formatos de arquivamento de pacotes compatíveis com operações ZIP padrão.

Você pode adicionar arquivos ao seu arquivo de teste antes de enviá-lo AWS Device Farm usando o seguinte comando:

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

Para um diretório de arquivos extras:

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

Esses comandos funcionam conforme o esperado para todos os formatos de upload de pacotes de teste, exceto para arquivos IPA. Para arquivos IPA, especialmente quando usados com XCUITests, recomendamos que você coloque os arquivos extras em um local um pouco diferente devido à forma como os pacotes de teste do AWS Device Farm iOS são resignados. Ao criar seu teste do iOS, o diretório do aplicativo de teste estará localizado dentro de outro diretório chamado*Payload*.

Por exemplo, é assim que um desses diretórios de teste do iOS pode parecer:

```
$ 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
```

Para esses XCUITest pacotes, adicione qualquer arquivo extra ao diretório que termina *`.app`* dentro do *`Payload`* diretório. Por exemplo, os comandos a seguir mostram como você pode adicionar um arquivo a esse pacote de teste:

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

Ao adicionar um arquivo ao seu pacote de teste, você pode esperar um comportamento de interação um pouco diferente no AWS Device Farm com base no formato de upload. Se o upload usou a extensão de arquivo ZIP, AWS Device Farm descompactará automaticamente o upload antes do teste e deixará os arquivos descompactados no local com a *`$DEVICEFARM_TEST_PACKAGE_PATH`* variável de ambiente. (Isso significa que se você adicionasse

um arquivo chamado *extra_file* à raiz do arquivo, como no primeiro exemplo, ele estaria localizado *$DEVICEFARM_TEST_PACKAGE_PATH/extra_file* durante o teste).

Para usar um exemplo mais prático, se você for um usuário do Appium TestNG que deseja incluir um *testng.xml* arquivo com seu teste, você pode incluí-lo em seu arquivo usando o seguinte comando:

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

Em seguida, você pode alterar seu comando de teste no modo de ambiente personalizado para o seguinte:

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

Se a extensão de upload do pacote de teste não for ZIP (por exemplo, arquivo APK, IPA ou JAR), o arquivo do pacote enviado em si será encontrado em*$DEVICEFARM_TEST_PACKAGE_PATH*. Como esses ainda são arquivos em formato de arquivamento, você pode descompactar o arquivo para acessar os arquivos adicionais de dentro. Por exemplo, o comando a seguir descompactará o conteúdo do pacote de teste (para arquivos APK, IPA ou JAR) no */tmp* diretório:

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

No caso de um arquivo APK ou JAR, você encontraria seus arquivos extras descompactados no */tmp* diretório (por exemplo,*/tmp/extra_file*). No caso de um arquivo IPA, conforme explicado anteriormente, os arquivos extras estariam em um local ligeiramente diferente dentro da pasta que termina em*.app*, que está dentro do *Payload* diretório. Por exemplo, com base no exemplo do IPA acima, o arquivo seria encontrado no local */tmp/Payload/ADFiOSReferenceAppUITests-Runner.app/extra_file* (referenciável como). */tmp/Payload/*.app/extra_file*

Para obter mais maneiras de ampliar sua suíte de testes e otimizar seus testes, consulteExtensão de ambientes de teste personalizados no Device Farm.

Acesso remoto no AWS Device Farm

O acesso remoto permite que você deslize o dedo, faça gestos e interaja com um dispositivo por meio de um navegador da web em tempo real para testar a funcionalidade e reproduzir os problemas dos clientes. Você interage com um dispositivo específico criando uma sessão de acesso remoto com esse dispositivo.

Uma sessão no Device Farm é uma interação em tempo real com um dispositivo físico real hospedado em um navegador da web. Uma sessão exibe o dispositivo específico que você selecionou ao iniciar a sessão. Um usuário pode iniciar mais de uma sessão por vez, mas o número total de dispositivos simultâneos está restrito ao número de slots para dispositivo que você tem. Você pode comprar slots de dispositivos com base na família de dispositivos (dispositivos Android ou iOS). Para obter mais informações, consulte Definição de preço do Device Farm.

Atualmente, o Device Farm oferece um subconjunto de dispositivos para testes de acesso remoto. Novos dispositivos são adicionados ao grupo de dispositivos sempre.

O Device Farm captura vídeo de cada sessão de acesso remoto e gera logs de atividade durante a sessão. Esses resultados incluem todas as informações que você fornece durante uma sessão.

> **Note**
>
> Por motivos de segurança, recomendamos evitar fornecer ou inserir informações confidenciais, como números de conta, informações pessoais de login e outros detalhes durante uma sessão de acesso remoto. Se possível, use alternativas desenvolvidas especificamente para testes, como contas de teste.

Tópicos

- Criação de uma sessão de acesso remoto no AWS Device Farm
- Usando uma sessão de acesso remoto no AWS Device Farm
- Recuperação dos resultados de uma sessão de acesso remoto no AWS Device Farm

Criação de uma sessão de acesso remoto no AWS Device Farm

Para obter informações sobre sessões de acesso remoto, consulte Sessões.

- Pré-requisitos
- Criar uma execução de teste (console)
- Próximas etapas

Pré-requisitos

- Crie um projeto no Device Farm. Siga as instruções em Criação de um projeto no AWS Device Farm e retorne para esta página.

Crie uma sessão com o console do Device Farm

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e, em seguida, Projetos.
3. Se você já tiver um projeto, selecione-o na lista. Caso contrário, crie um projeto seguindo as instruções em Criação de um projeto no AWS Device Farm.
4. Na guia Remote access (Acesso remoto), escolha Start a new session (Iniciar uma nova sessão).
5. Escolha um dispositivo para sua sessão. Você pode escolher na lista de dispositivos disponíveis ou pesquisar um dispositivo usando a barra de pesquisa na parte superior da lista. Você pode pesquisar por:
   - Name
   - Plataforma
   - Formato
   - Tipo de frota
6. Em Session name (Nome da sessão), insira um nome para a sessão.
7. Escolha Confirm and start session (Confirmar e iniciar sessão).

Próximas etapas

O Device Farm inicia a sessão assim que o dispositivo solicitado estiver disponível, normalmente em alguns minutos. A caixa de diálogo Dispositivo solicitado ficará aberta até o momento em que a sessão iniciar. Para cancelar a solicitação de sessão, escolha Cancel request (Cancelar solicitação).

Depois que a sessão for iniciada, se tiver que fechar o navegador ou a guia do navegador sem interromper a sessão ou se a conexão entre o navegador e a internet se perder, a sessão permanecerá ativa por cinco minutos. Depois disso, o Device Farm encerra a sessão. A conta é cobrada pelo tempo ocioso.

Depois que a sessão for iniciada, você poderá interagir com o dispositivo no navegador da web.

Usando uma sessão de acesso remoto no AWS Device Farm

Para obter informações sobre como executar testes interativos de aplicativos Android e iOS por meio de sessões de acesso remoto, consulte Sessões.

- Pré-requisitos
- Use uma sessão no console do Device Farm
- Próximas etapas
- Dicas e truques

Pré-requisitos

- Crie uma sessão. Siga as instruções em Criar uma sessão e retorne para esta página.

Use uma sessão no console do Device Farm

Assim que o dispositivo que você solicitou para uma sessão de acesso remoto ficar disponível, o console exibirá a tela do dispositivo. A sessão tem duração máxima de 150 minutos. O tempo restante na sessão aparece no campo Tempo restante próximo ao nome do dispositivo.

Instalar um aplicativo

Para instalar uma aplicação no dispositivo de sessão, em Instalar aplicações, selecione Escolher arquivo e escolha o arquivo .apk (Android) ou o arquivo .ipa (iOS) que você deseja instalar. Os aplicativos que você executa em uma sessão de acesso remoto não exigem nenhum teste de instrumentação nem provisionamento.

> **Note**
>
> O AWS Device Farm não exibe uma confirmação depois que um aplicativo é instalado. Experimente interagir com o aplicativo para ver se ele já pode ser usado.

Ao fazer upload de um aplicativo, às vezes o aplicativo demora um pouco para ficar disponível. Examine a bandeja do sistema para determinar se o aplicativo está disponível.

Controlar o dispositivo

Você pode interagir com o dispositivo exibido no console assim como faria com um dispositivo físico real, usando o mouse ou um dispositivo semelhante para tocar e o teclado na tela do dispositivo. Para dispositivos Android, existem botões em View controls (Visualizar controles) que funcionam como os botões Home (Início) e Back (Voltar) em um dispositivo Android. Para dispositivos iOS, existe um botão Home (Início) que funciona da mesma forma que o botão de início em um dispositivo iOS. Você também pode alternar aplicações executadas no dispositivo escolhendo Aplicações recentes.

Alternar modos retrato e paisagem

Você também pode alternar os modos retrato (vertical) e paisagem (horizontal) nos dispositivos que está usando.

Próximas etapas

O Device Farm continua a sessão até que você a interrompa manualmente ou até que o limite de tempo de 150 minutos seja atingido. Para encerrar a sessão, escolha o botão Interromper sessão. Depois que a sessão for interrompida, você poderá acessar o vídeo que foi capturado e os logs que foram gerados. Para obter mais informações, consulte Recuperando resultados da sessão de sessões de acesso remoto.

Dicas e truques

Você pode ter problemas de desempenho com a sessão de acesso remoto em algumas AWS regiões. Em parte, isso se deve à latência em algumas regiões. Se tiver problemas de desempenho, permita que a sessão de acesso remota recupere o atraso para então interagir novamente com o aplicativo.

Recuperação dos resultados de uma sessão de acesso remoto no AWS Device Farm

Para obter informações sobre sessões, consulte Sessões.

- Pré-requisitos
- Visualização de detalhes da sessão
- Download de vídeo ou logs de sessão

Pré-requisitos

- Conclua uma sessão. Siga as instruções em Usando uma sessão de acesso remoto no AWS Device Farm e retorne para esta página.

Visualização de detalhes da sessão

Quando uma sessão de acesso remoto termina, o console do Device Farm exibe uma tabela que contém detalhes sobre a atividade durante a sessão. Para obter mais informações, consulte Analisar informações do log.

Para retornar aos detalhes de uma sessão em um momento posterior:

1. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e, em seguida, Projetos.
2. Escolha o projeto que contém a sessão.
3. Selecione Acesso remoto e escolha a sessão que você deseja revisar na lista.

Download de vídeo ou logs de sessão

Quando uma sessão de acesso remoto termina, o console do Device Farm fornece acesso a uma captura de vídeo da sessão e aos logs de atividade. Nos resultados da sessão, escolha a guia Files (Arquivos) para obter uma lista de links para o vídeo e os logs da sessão. Você pode visualizar esses arquivos no navegador ou os salvar localmente.

Dispositivos privados no AWS Device Farm

Um dispositivo privado é um dispositivo móvel físico que o AWS Device Farm implementa em seu nome em um data center da Amazon. Este dispositivo é exclusivo para sua AWS conta.

> **Note**
>
> Atualmente, os dispositivos privados estão disponíveis somente na região Oeste AWS dos EUA (Oregon) (`us-west-2`).

Se tiver uma frota de dispositivos privados, você pode criar sessões de acesso remoto e programar execuções de teste usando seus dispositivos privados. Para obter mais informações, consulte Criar uma execução de teste ou iniciar uma sessão de acesso remoto no AWS Device Farm. Você também pode criar perfis de instância para controlar o comportamento dos seus dispositivos privados durante uma sessão de acesso remoto ou a execução de um teste. Para obter mais informações, consulte Criação de um perfil de instância no AWS Device Farm. Opcionalmente, você pode solicitar que determinados dispositivos Android privados sejam implantados como dispositivos rooteados.

Você também pode criar um serviço de endpoint de nuvem privada virtual da Amazon para testar aplicações privadas às quais sua empresa tem acesso, mas que não podem ser acessados pela Internet. Por exemplo, você pode ter um aplicativo web em execução dentro da sua VPC que você deseja testar em dispositivos móveis. Para obter mais informações, consulte Usando serviços de endpoint da Amazon VPC com Device Farm - Legacy (não recomendado).

Se você tiver interesse em usar uma frota de dispositivos privados, fale conosco. A equipe do Device Farm deve trabalhar com você para configurar e implantar uma frota de dispositivos privados AWS em sua conta.

Tópicos

- Criação de um perfil de instância no AWS Device Farm
- Solicite dispositivos privados adicionais no AWS Device Farm
- Criar uma execução de teste ou iniciar uma sessão de acesso remoto no AWS Device Farm
- Seleção de dispositivos privados em um pool de dispositivos no AWS Device Farm
- Ignorar a nova assinatura de aplicações em dispositivos privados no AWS Device Farm
- Amazon VPC em todas AWS as regiões no AWS Device Farm

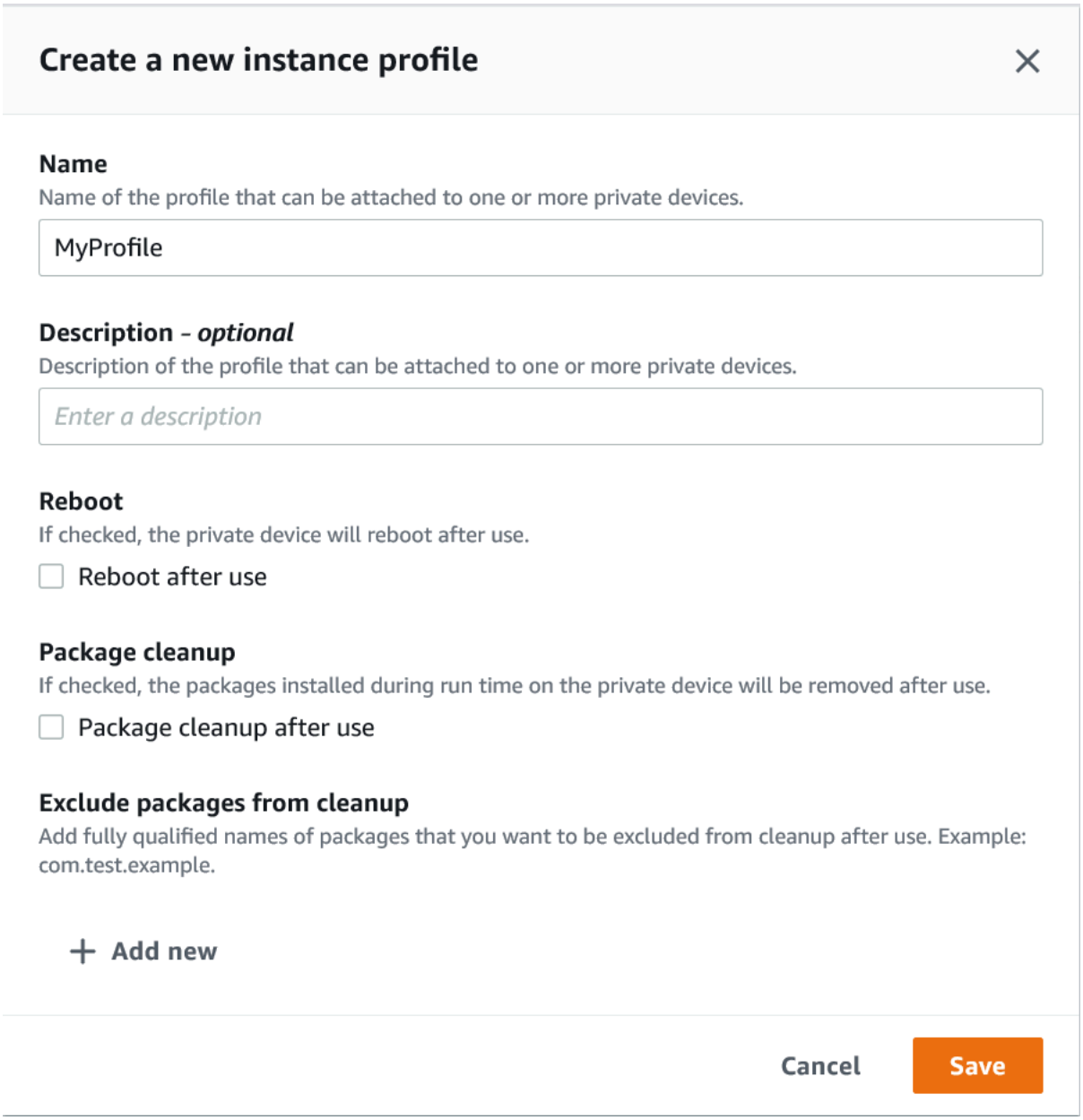
- Encerramento de dispositivos privados no Device Farm

Criação de um perfil de instância no AWS Device Farm

Você pode configurar uma frota que contém um ou mais dispositivos privados. Esses dispositivos são dedicados à sua conta do AWS . Depois de configurar os dispositivos, você pode opcionalmente criar um ou mais perfis de instância para eles. Perfis de instância podem ajudar você a automatizar execuções de teste e consistentemente aplicar as mesmas configurações para instâncias do dispositivo. Os perfis de instâncias também podem ajudar você a controlar o comportamento da sessão de acesso remoto. Para obter mais informações sobre dispositivos privados no Device Farm, consulteDispositivos privados no AWS Device Farm.

Para criar uma instância

1. Abra o console do Device Farm em https://console.aws.amazon.com/devicefarm/.
2. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e Dispositivos privados.
3. Escolha Perfis de instância.
4. Escolha Criar perfil de instância.
5. Insira um nome para o perfil de instância.



6. (Opcional) Digite uma descrição do perfil de instância.
7. (Opcional) Altere qualquer uma das seguintes configurações para especificar quais ações você deseja que o Device Farm execute em um dispositivo após o término de cada execução de teste ou sessão:

   - Reinicia após o uso: para reiniciar o dispositivo, marque essa caixa de seleção. Por padrão, essa caixa de seleção fica desmarcada (`false`).
   - Limpeza do pacote: para remover todos os pacotes de aplicações que você instalou no dispositivo, marque essa caixa de seleção. Por padrão, essa caixa de seleção fica

desmarcada (`false`). Para manter todos os pacotes de aplicativos que você instalou no dispositivo, deixe esta caixa de seleção desmarcada.

- Excluir pacotes da limpeza: para manter apenas pacotes selecionados de aplicações no dispositivo, marque a caixa de seleção Limpeza do pacote e escolha Adicionar novo. Para o nome do pacote, insira o nome totalmente qualificado do pacote de aplicativos que você deseja manter no dispositivo (por exemplo, `com.test.example`). Para manter mais pacotes de aplicações no dispositivo, escolha Adicionar novo e, em seguida, digite o nome totalmente qualificado de cada pacote.

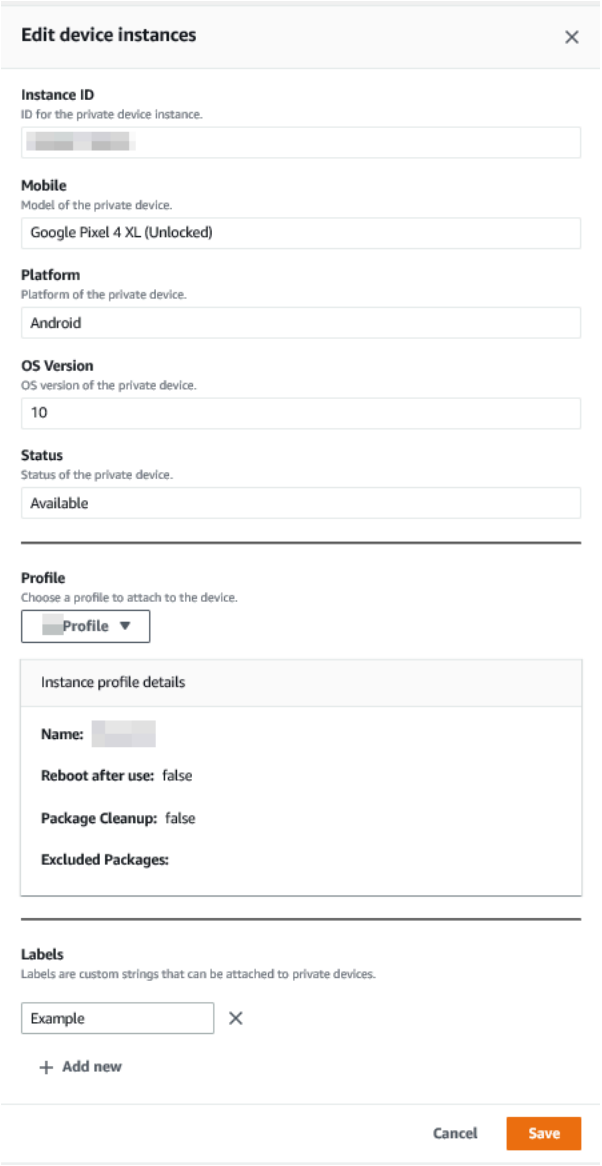
8. Escolha Salvar.

Solicite dispositivos privados adicionais no AWS Device Farm

No AWS Device Farm, você pode solicitar que instâncias adicionais de dispositivos privados sejam adicionadas à sua frota. Você também pode visualizar e alterar as configurações das instâncias de dispositivos privados existentes em sua frota. Para obter mais informações sobre dispositivos privados, consulteDispositivos privados no AWS Device Farm.

Para solicitar dispositivos privados adicionais ou alterar suas configurações

1. Abra o console do Device Farm em https://console.aws.amazon.com/devicefarm/.
2. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e Dispositivos privados.
3. Escolha Instâncias do dispositivo. A guia Instâncias do dispositivo exibe uma tabela dos dispositivos privados em sua frota. Para pesquisar ou filtrar rapidamente a tabela, insira os termos de pesquisa na barra de pesquisa acima das colunas.
4. Para solicitar uma nova instância de dispositivo privado, escolha Solicitar instância de dispositivo ou entre em contato conosco. Os dispositivos privados exigem configuração adicional com a ajuda da equipe do Device Farm.
5. Na tabela de instâncias de dispositivos, selecione a opção de alternância ao lado da instância sobre a qual você deseja visualizar informações ou gerenciar e selecione Editar.



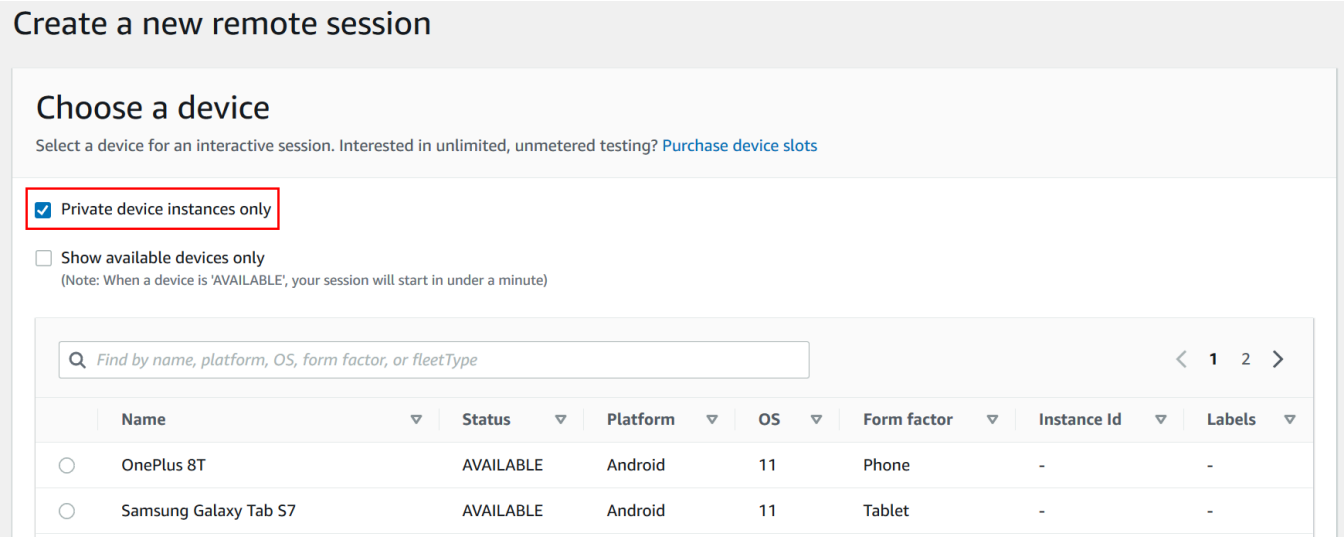
6. Para anexar um perfil de instância à instância do dispositivo, escolha-o na lista suspensa Perfil. Anexar um perfil de instância pode ser útil se você quiser sempre excluir um pacote de aplicativo específico das tarefas de limpeza, por exemplo. Para obter mais informações sobre o uso de perfis de instância com dispositivos, consulteCriação de um perfil de instância no AWS Device Farm.
7. (Opcional) Em Rótulos, escolha Adicionar novo para adicionar um rótulo à instância do dispositivo. Os rótulos podem ajudar você a categorizar seus dispositivos e encontrar dispositivos específicos com mais facilidade.
8. Escolha Salvar.

Criar uma execução de teste ou iniciar uma sessão de acesso remoto no AWS Device Farm

No AWS Device Farm, depois de configurar uma frota de dispositivos privados, você pode criar execuções de teste ou iniciar sessões de acesso remoto com um ou mais dispositivos privados em sua frota. Para obter mais informações sobre dispositivos privados, consulteDispositivos privados no AWS Device Farm.

Para criar uma execução de teste ou iniciar uma sessão de acesso remoto

1. Abra o console do Device Farm em https://console.aws.amazon.com/devicefarm/.
2. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e, em seguida, Projetos.
3. Escolha um projeto existente na lista ou crie um novo. Para criar um projeto, selecione Novo projeto, insira um nome para o projeto e selecione Enviar.
4. Execute um destes procedimentos:
   - Para criar uma execução de teste, escolha Testes automatizados e, em seguida, Criar nova execução. O assistente orienta você durante as etapas para criação da execução. Na etapa Selecionar dispositivos, você pode editar um pool de dispositivos existente ou criar um novo pool de dispositivos que inclua somente os dispositivos privados que a equipe do Device Farm configurou e associou à sua AWS conta. Para obter mais informações, consulte the section called “Criar um grupo de dispositivos privados”.
   - Para iniciar uma sessão de acesso remoto, escolha Acesso remoro, em seguida, Iniciar nova sessão. Na página Escolha um dispositivo, selecione Somente instâncias privadas de dispositivos para limitar a lista somente aos dispositivos privados que a equipe do Device Farm configurou e associou à sua AWS conta. Em seguida, escolha o dispositivo que você deseja acessar, insira um nome para a sessão de acesso remoto e escolha Confirmar e iniciar a sessão.



Seleção de dispositivos privados em um pool de dispositivos no AWS Device Farm

Para usar dispositivos privados em sua execução de teste, você pode criar um pool de dispositivos que seleciona seus dispositivos privados. Os pools de dispositivos permitem que você selecione dispositivos privados principalmente por meio de três tipos de regras de pool de dispositivos:

1. Regras com base no ARN do dispositivo
2. Regras com base no rótulo da instância do dispositivo
3. Regras com base no ARN da instância do dispositivo

Nas seções a seguir, cada tipo de regra e seus casos de uso são descritos detalhadamente. Você pode usar o console Device Farm, a Interface de Linha de AWS Comando (AWS CLI) ou a API Device Farm para criar ou modificar um pool de dispositivos com dispositivos privados usando essas regras.

Tópicos

- ARN do dispositivo
- Rótulos de instância do dispositivo
- Instância ARN
- Criação de um pool de dispositivos privados com dispositivos privados (console)
- Criação de um grupo de dispositivos privados com dispositivos privados (AWS CLI)

- Criação de um pool de dispositivos privados com dispositivos privados (API)

ARN do dispositivo

O ARN de um dispositivo é um identificador que representa um tipo de dispositivo em vez de qualquer instância específica de dispositivo físico. Um tipo de dispositivo é definido pelos seguintes atributos:

- O ID da frota do dispositivo
- O OEM do dispositivo
- O número do modelo do dispositivo
- A versão do sistema operacional do dispositivo.
- O estado do dispositivo que indica se ele está enraizado ou não

Muitas instâncias de dispositivos físicos podem ser representadas por um único tipo de dispositivo, em que cada instância desse tipo tem os mesmos valores para esses atributos. Por exemplo, se você tivesse três `Apple iPhone 13` dispositivos na versão iOS `16.1.0` em sua frota privada, cada dispositivo compartilharia o mesmo ARN do dispositivo. Se algum dispositivo fosse adicionado ou removido da sua frota com esses mesmos atributos, o ARN do dispositivo continuaria a representar quaisquer dispositivos disponíveis que você tivesse em sua frota para esse tipo de dispositivo.

O ARN do dispositivo é a maneira mais robusta de selecionar dispositivos privados para um pool de dispositivos, pois permite que o pool de dispositivos continue selecionando dispositivos, independentemente das instâncias de dispositivos específicas que você implantou a qualquer momento. Instâncias individuais de dispositivos privados podem apresentar falhas de hardware, fazendo com que o Device Farm as substitua automaticamente por novas instâncias funcionais do mesmo tipo de dispositivo. Nesses cenários, a regra de ARN do dispositivo garante que seu pool de dispositivos possa continuar selecionando dispositivos no caso de uma falha de hardware.

Quando você usa uma regra de ARN de dispositivo para dispositivos privados em seu pool de dispositivos e agenda uma execução de teste com esse pool, o Device Farm verifica automaticamente quais instâncias de dispositivos privados são representadas pelo ARN desse dispositivo. Das instâncias que estão disponíveis atualmente, uma delas será designada para executar seu teste. Se nenhuma instância estiver disponível no momento, o Device Farm aguardará até que a primeira instância disponível do ARN desse dispositivo fique disponível e a atribuirá para executar seu teste.

Rótulos de instância do dispositivo

Um rótulo de instância de dispositivo é um identificador textual que você pode anexar como metadados para uma instância de dispositivo. Você pode anexar vários rótulos a cada instância do dispositivo e o mesmo rótulo a várias instâncias do dispositivo. Para obter mais informações sobre como adicionar, modificar ou remover rótulos de dispositivos de instâncias de dispositivos, consulte Managing private devices.

O rótulo da instância do dispositivo pode ser uma forma robusta de selecionar dispositivos privados para um pool de dispositivos porque, se você tiver várias instâncias de dispositivos com o mesmo rótulo, ele permitirá que o pool de dispositivos selecione qualquer uma delas para seu teste. Se o ARN do dispositivo não for uma boa regra para seu caso de uso (por exemplo, se você quiser selecionar entre dispositivos de vários tipos de dispositivos ou se quiser selecionar entre um subconjunto de todos os dispositivos de um tipo de dispositivo), os rótulos de instância do dispositivo poderão permitir que você selecione entre vários dispositivos para seu pool de dispositivos com maior granularidade. Instâncias individuais de dispositivos privados podem apresentar falhas de hardware, fazendo com que o Device Farm as substitua automaticamente por novas instâncias funcionais do mesmo tipo de dispositivo. Nesses cenários, a instância do dispositivo substituto não reterá nenhum metadado do rótulo da instância do dispositivo substituído. Portanto, se você aplicar o mesmo rótulo de instância de dispositivo a várias instâncias de dispositivos, a regra de rótulo de instância de dispositivo garantirá que seu pool de dispositivos possa continuar selecionando instâncias de dispositivos no caso de uma falha de hardware.

Quando você usa uma regra de rótulo de instância de dispositivo para dispositivos privados em seu pool de dispositivos e agenda uma execução de teste com esse pool, o Device Farm verifica automaticamente quais instâncias de dispositivos particulares são representadas por esse rótulo de instância de dispositivo e, dessas instâncias, seleciona aleatoriamente uma que esteja disponível para executar seu teste. Se nenhuma estiver disponível, o Device Farm selecionará aleatoriamente qualquer instância de dispositivo com o rótulo de instância do dispositivo para executar seu teste e colocará o teste em fila para execução no dispositivo quando estiver disponível.

Instância ARN

O ARN de uma instância de dispositivo é um identificador que representa uma instância física de dispositivo bare metal implantada em uma frota privada. Por exemplo, se você tivesse três `iPhone 13` dispositivos no sistema operacional `15.0.0` em sua frota privada, enquanto cada dispositivo compartilhasse o mesmo ARN do dispositivo, cada dispositivo também teria seu próprio ARN de instância representando apenas essa instância.

O ARN da instância do dispositivo é a maneira menos eficiente de selecionar dispositivos privados para um pool de dispositivos e só é recomendado se os rótulos do dispositivo ARNs e da instância do dispositivo não se adequarem ao seu caso de uso. As instâncias de dispositivos geralmente ARNs são usadas como regras para grupos de dispositivos quando uma instância de dispositivo específica é configurada de forma única e específica como um pré-requisito para seu teste e se essa configuração precisa ser conhecida e verificada antes que o teste seja executado nela. Instâncias individuais de dispositivos privados podem apresentar falhas de hardware, fazendo com que o Device Farm as substitua automaticamente por novas instâncias funcionais do mesmo tipo de dispositivo. Nesses cenários, a instância do dispositivo substituto terá um ARN de instância de dispositivo diferente do dispositivo substituído. Portanto, se você depende da instância de dispositivos ARNs para seu pool de dispositivos, precisará alterar manualmente a definição da regra do pool de dispositivos de usar o ARN antigo para usar o novo ARN. Se você precisar pré-configurar manualmente o dispositivo para o teste, esse pode ser um fluxo de trabalho eficaz (comparado ao dispositivo ARNs). Para testes em grande escala, é recomendável tentar adaptar esses casos de uso para trabalhar com rótulos de instância de dispositivos e, se possível, ter várias instâncias de dispositivos pré-configuradas para testes.

Quando você usa uma regra ARN de instância de dispositivo para dispositivos privados em seu pool de dispositivos e agenda uma execução de teste com esse pool, o Device Farm atribui automaticamente esse teste a essa instância de dispositivo. Se essa instância do dispositivo não estiver disponível, o Device Farm colocará o teste em fila no dispositivo assim que ele estiver disponível.

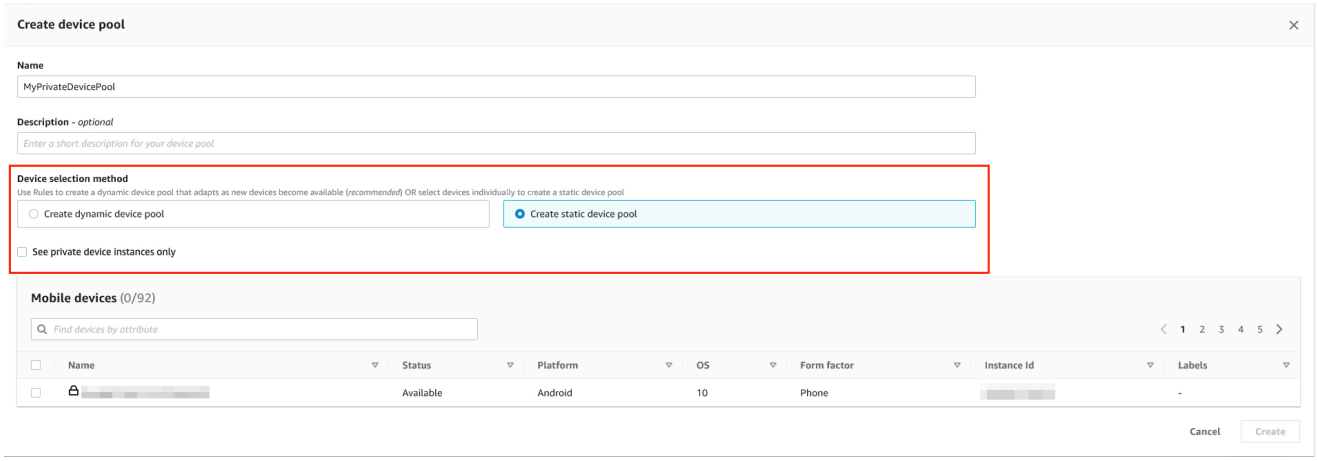
Criação de um pool de dispositivos privados com dispositivos privados (console)

Ao criar uma execução de teste, você pode criar um grupo de dispositivos para a execução de teste e garantir que o grupo inclua apenas os dispositivos privados.

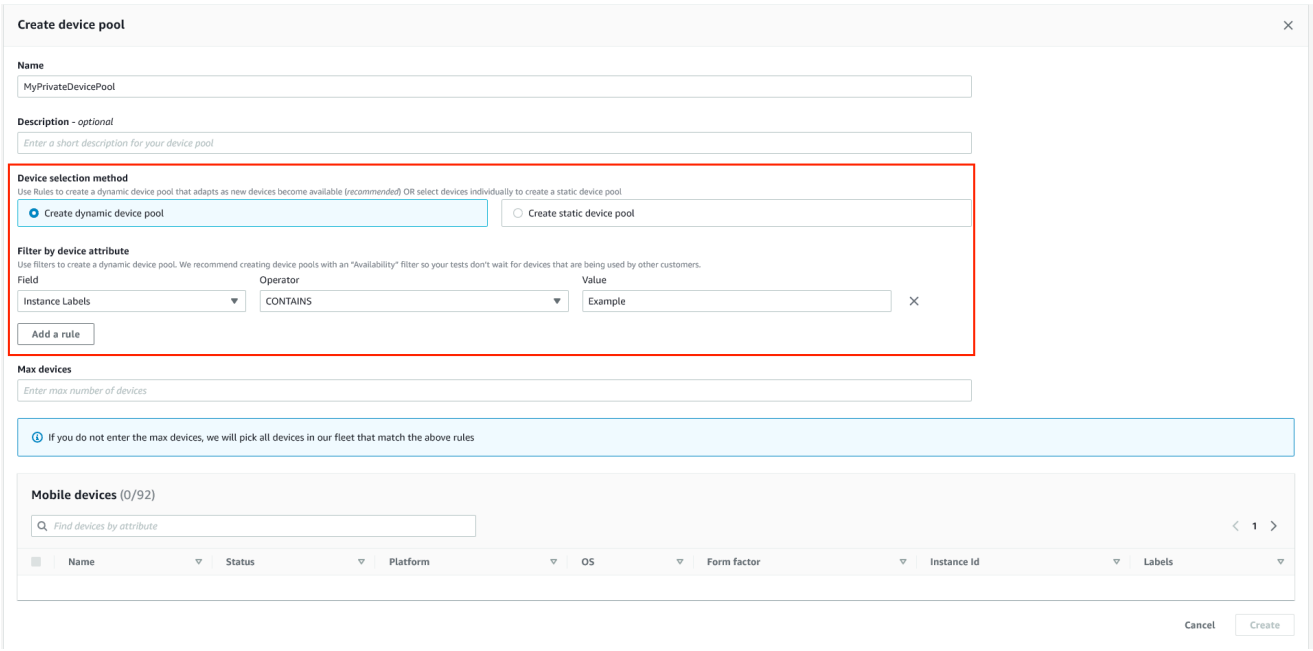
> **Note**
>
> Ao criar um pool de dispositivos com dispositivos privados no console, você só pode usar qualquer uma das três regras disponíveis para selecionar dispositivos privados. Se você quiser criar um pool de dispositivos que contenha vários tipos de regras para dispositivos privados (por exemplo, pools de dispositivos que contêm regras para dispositivos ARNs e instâncias de dispositivos ARNs), é necessário criar o pool por meio da CLI ou da API.

1. Abra o console do Device Farm em https://console.aws.amazon.com/devicefarm/.
2. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e, em seguida, Projetos.
3. Escolha um projeto existente na lista ou crie um novo. Para criar um projeto, selecione Novo projeto, insira um nome para o projeto e selecione Enviar.
4. Escolha Configurações do projeto e, em seguida, navegue até a guia Pools de dispositivos.
5. Escolha Criar pool de dispositivos e insira um nome e uma descrição opcional para seu pool de dispositivos.

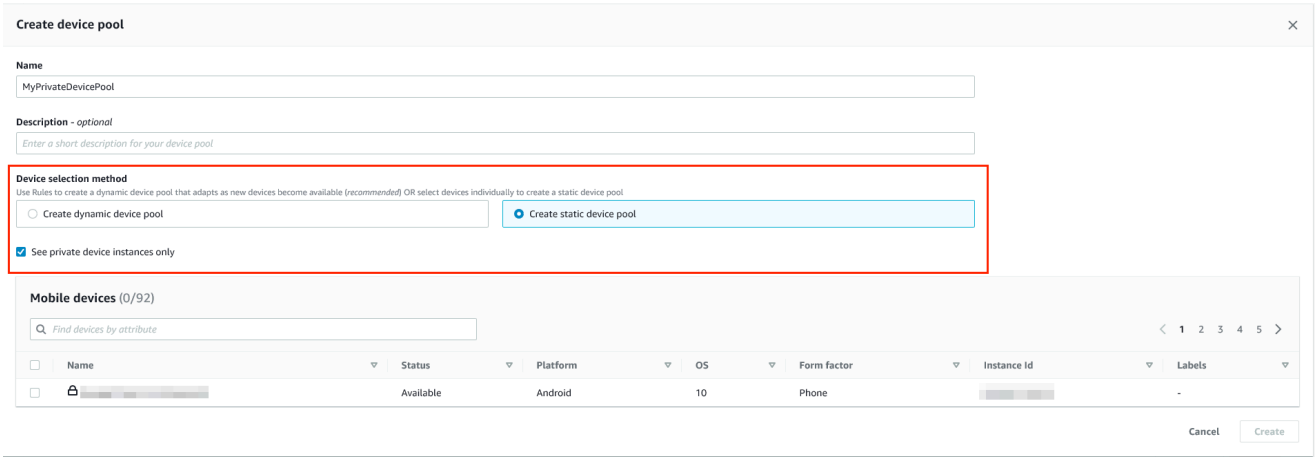
   a. Para usar regras de ARN de dispositivos para seu grupo de dispositivos, escolha Criar grupos estáticos de dispositivos e selecione os tipos de dispositivos específicos da lista que você gostaria de usar no grupo de dispositivos. Não selecione Somente instâncias de dispositivos privados porque essa opção faz com que o grupo de dispositivos seja criado com regras de ARN da instância de dispositivo (em vez de regras de ARN de dispositivo).



   b. Para usar regras de rótulo de instância de dispositivo para o grupo de dispositivos, escolha Criar grupos dinâmicos de dispositivos. Em seguida, para cada rótulo que você gostaria de usar no grupo de dispositivos, escolha Adicionar uma regra. Para cada regra, escolha Rótulos de instância como `Field`, escolha Contém como `Operator` e especifique o rótulo de instância do dispositivo desejado como `Value`.



c. Para usar regras de ARN de instância de dispositivo para seu pool de dispositivos, escolha Create static device pool e selecione Private device instances only para limitar a lista de dispositivos somente às instâncias privadas que o Device Farm associou à sua AWS conta.



6. Escolha Criar.

Criação de um grupo de dispositivos privados com dispositivos privados (AWS CLI)

- Execute o comando create-device-pool.

Para obter informações sobre como usar o Device Farm com o AWS CLI, consulteAWS CLI referência.

Criação de um pool de dispositivos privados com dispositivos privados (API)

- Chame a API `CreateDevicePool`.

Para obter informações sobre como usar a API do Device Farm, consulte Automatização do Device Farm.

Ignorar a nova assinatura de aplicações em dispositivos privados no AWS Device Farm

A assinatura de aplicativos é um processo que envolve a assinatura digital de um pacote de aplicativos (por exemplo, APK, IPA) com uma chave privada antes que ele possa ser instalado em um dispositivo ou publicado em uma loja de aplicativos como a Google Play Store ou a Apple App Store. Para simplificar os testes reduzindo o número de assinaturas e perfis necessários e aumentar a segurança dos dados em dispositivos remotos, o AWS Device Farm assinará novamente seu aplicativo após o upload para o serviço.

Depois de fazer o upload do seu aplicativo para o AWS Device Farm, o serviço gerará uma nova assinatura para o aplicativo usando seus próprios certificados de assinatura e perfis de provisionamento. Esse processo substitui a assinatura original do aplicativo pela assinatura do AWS Device Farm. O aplicativo reassinado é então instalado nos dispositivos de teste fornecidos pelo AWS Device Farm. A nova assinatura permite que o aplicativo seja instalado e executado nesses dispositivos sem a necessidade dos certificados originais do desenvolvedor.

No iOS, substituímos o perfil de provisionamento incorporado por um perfil curinga e renunciamos ao aplicativo. Se você os fornecer, adicionaremos dados auxiliares ao pacote do aplicativo antes da instalação para que os dados estejam presentes na sandbox do seu aplicativo. A renúncia do aplicativo iOS resulta na remoção de determinados direitos. Isso inclui grupo de aplicativos, domínios associados, Game Center,, HealthKit HomeKit, Configuração de acessórios sem fio, compra no aplicativo, áudio entre aplicativos, Apple Pay, notificações push e configuração e controle de VPN.

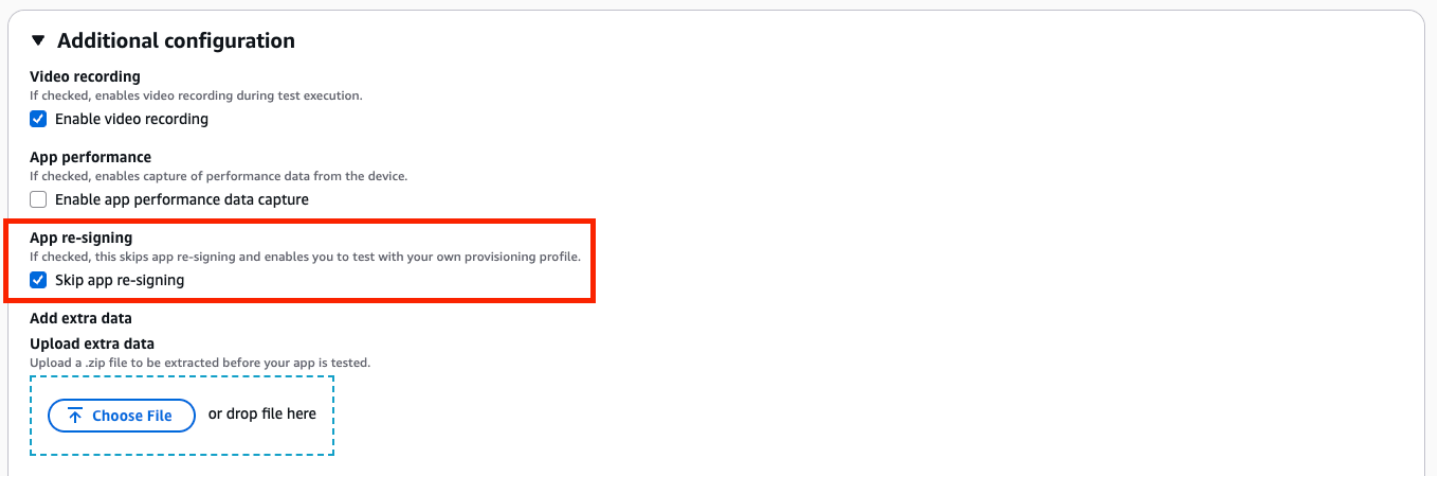
No Android, renunciamos ao aplicativo. Isso pode interromper a funcionalidade que depende da assinatura do aplicativo, como a API Android do Google Maps. Também pode acionar a detecção

antipirataria e antiadulteração disponível em produtos como. DexGuard Para testes integrados, podemos modificar o manifesto para incluir as permissões necessárias para capturar e salvar capturas de tela.

Quando você usa dispositivos privados, pode ignorar a etapa em que o AWS Device Farm assina novamente a aplicação. Isso é diferente dos dispositivos públicos, em que o Device Farm sempre assina novamente a aplicação nas plataformas Android e iOS.

Você pode ignorar a nova assinatura do aplicativo ao criar uma sessão de acesso remoto ou a execução de um teste. Isso poderá ser útil se a aplicação tiver uma funcionalidade que seja interrompida quando o Device Farm assinar novamente a aplicação. Por exemplo, notificações por push podem não funcionar depois da nova assinatura. Para obter mais informações sobre as alterações que o Device Farm faz ao testar seu aplicativo, consulte o AWS Device Farm FAQs ou a página Apps.

Para ignorar a reassinatura do aplicativo para um teste, selecione Ignorar a reassinatura do aplicativo em Configuração adicional. Essa opção está disponível somente para dispositivos privados.



> **Note**
>
> Se você estiver usando a XCTest estrutura, a opção Ignorar a reassinatura do aplicativo não estará disponível. Para obter mais informações, consulte Integrando o Device Farm com XCTest o iOS.

Etapas adicionais para configurar suas configurações de assinatura de aplicativos variam, dependendo de você estar usando dispositivos privados Android ou iOS.

Ignorar a nova assinatura da aplicação em dispositivos Android

Se você estiver testando o aplicativo em um dispositivo Android privado, selecione Skip app re-signing (Ignorar nova assinatura do aplicativo) ao criar a execução de teste ou a sessão de acesso remoto. Nenhuma outra configuração é necessária.

Ignorar a nova assinatura da aplicação em dispositivos iOS

A Apple exige que você assine um aplicativo para teste antes de carregá-lo em um dispositivo. Para dispositivos iOS, você tem duas opções para assinar seu aplicativo.

- Se estiver usando um perfil de desenvolvedor interno (Enterprise), você pode passar para a próxima seção, the section called “Criar uma sessão de acesso remoto para confiar na sua aplicação”.

- Se você estiver usando um perfil de desenvolvimento de aplicativo iOS ad hoc, você deverá primeiramente registrar o dispositivo com a conta de desenvolvedor da Apple e atualizar o perfil de provisionamento para incluir o dispositivo privado. Você deve assinar novamente o aplicativo com o perfil de provisionamento que você atualizou. Você pode executar a aplicação assinada novamente no Device Farm.

Para registrar um dispositivo com um perfil de aprovisionamento de desenvolvimento de aplicativos iOS ad hoc

1. Faça login em sua conta de desenvolvedor da Apple.
2. Navegue até a seção IDsCertificados e Perfis do console.
3. Vá até Devices (Dispositivos).
4. Registre o dispositivo em sua conta de desenvolvedor da Apple. Para obter o nome e o UDID do dispositivo, use a operação `ListDeviceInstances` da API do Device Farm.
5. Acesse seu perfil de provisionamento e escolha Editar.
6. Selecione o dispositivo na lista.
7. No XCode, busque seu perfil de provisionamento atualizado e, em seguida, assine novamente o aplicativo.

Nenhuma outra configuração é necessária. Agora, você pode criar uma sessão de acesso remoto ou uma execução de teste e selecionar Skip app re-signing (Ignorar nova assinatura do aplicativo).

Criar uma sessão de acesso remoto para confiar na sua aplicação para iOS

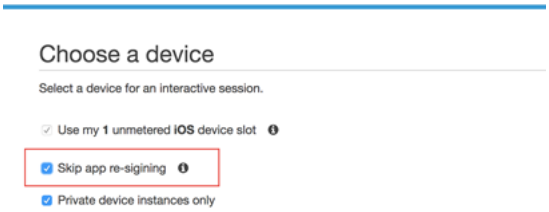
Se estiver usando um perfil de provisionamento de desenvolvedor interno (Enterprise), você precisará fazer um único procedimento para confiar no certificado de desenvolvedor do aplicativo interno em cada um dos dispositivos privados.

Para isso, você pode instalar o aplicativo que deseja testar no dispositivo privado ou instalar um aplicativo fictício assinado com o mesmo certificado do aplicativo que deseja testar. Há uma vantagem em instalar um aplicativo fictício assinado com o mesmo certificado. A vantagem é que, depois de confiar no perfil de configuração ou no desenvolvedor do aplicativo empresarial, todos os aplicativos desse desenvolvedor serão confiáveis no dispositivo privado até que você os exclua. Dessa forma, quando fizer upload de uma nova versão do aplicativo que deseja testar, você não precisa confiar no desenvolvedor de aplicativos novamente. Isso é especialmente útil se você executar as automações de teste e não quiser criar uma sessão de acesso remoto cada vez que testar seu aplicativo.

Antes de iniciar sua sessão de acesso remoto, siga as etapas em Criação de um perfil de instância no AWS Device Farm para criar ou modificar um perfil de instância no Device Farm. No perfil de instância, adicione o ID do pacote da aplicação de teste ou aplicação simulada à configuração Excluir pacotes da limpeza. Em seguida, anexe o perfil de instância à instância do dispositivo privado para garantir que o Device Farm não remova essa aplicação do dispositivo antes de iniciar uma nova execução de teste. Isso garante que o certificado do desenvolvedor permaneça confiável.

Você pode fazer upload do aplicativo fictício para o dispositivo usando uma sessão de acesso remoto, o que permite iniciar o aplicativo e confiar no desenvolvedor.

1. Siga as instruções em Criar uma sessão para criar uma sessão de acesso remoto usando o perfil da instância de dispositivo privado que você acabou de criar. Ao criar a sessão, não se esqueça de selecionar Skip app re-signing (Ignorar nova assinatura do aplicativo).



> **Important**
>
> Para filtrar a lista de dispositivos de modo a incluir apenas dispositivos privados, selecione Private device instances only (Somente instâncias de dispositivos privados) para garantir que você use um dispositivo privado com o perfil de instância correto.

Além disso, adicione a aplicação simulada ou a aplicação que você deseja testar à configuração Excluir pacotes da limpeza para o perfil de instância que está anexado a essa instância.

2. Quando a sessão remota for iniciada, escolha Escolher arquivo para instalar uma aplicação que use seu perfil de provisionamento interno.
3. Inicie o aplicativo recém-carregado.
4. Siga as instruções para confiar no certificado do desenvolvedor.

Todos os aplicativos desse perfil de configuração ou do desenvolvedor do aplicativo empresarial já são confiáveis nesse dispositivo privado até que você os exclua.

Amazon VPC em todas AWS as regiões no AWS Device Farm

Os serviços do Device Farm estão localizados apenas na região oeste dos EUA (Oregon) (`us-west-2`). Você pode usar a Amazon Virtual Private Cloud (Amazon VPC) para acessar um serviço em sua Amazon Virtual Private Cloud em outra AWS região usando o Device Farm. Se o Device Farm e seu serviço estiverem na mesma região, consulte Usando serviços de endpoint da Amazon VPC com Device Farm - Legacy (não recomendado).

Há duas maneiras de acessar seus serviços privados localizados em uma região diferente. Se você tiver serviços localizados em uma outra região que não seja `us-west-2`, poderá usar o emparelhamento de VPC para emparelhar a VPC dessa região com outra VPC que esteja fazendo interface com o Device Farm em `us-west-2`. No entanto, se você tiver serviços em várias regiões, um Transit Gateway permitirá que você acesse esses serviços com uma configuração de rede mais simples.

Para obter mais informações, consulte VPC peering scenarios no Guia de emparelhamento da Amazon VPC.

Visão geral do emparelhamento de VPC VPCs em diferentes regiões no AWS Device Farm

Você pode emparelhar quaisquer dois VPCs em regiões diferentes, desde que tenham blocos CIDR distintos e não sobrepostos. Isso garante que todos os endereços IP privados sejam exclusivos e permite que todos os recursos do se dirijam uns VPCs aos outros sem a necessidade de qualquer forma de tradução de endereços de rede (NAT). Para obter mais informações sobre notação de CIDR, consulte RFC 4632.

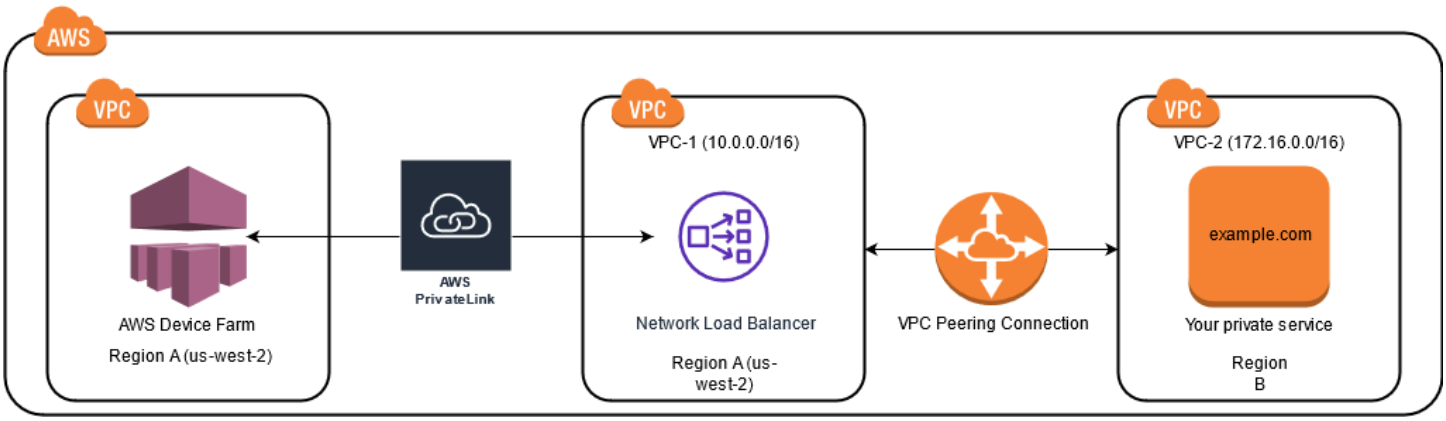
Este tópico inclui um cenário de exemplo entre regiões no qual o Device Farm (denominado VPC-1) está localizado na região Oeste dos EUA (Oregon) (`us-west-2`). A segunda VPC neste exemplo (chamada de VPC-2) está em outra região.

Exemplo de VPC entre regiões do Device Farm

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

> **Important**
>
> Estabelecer uma conexão de emparelhamento entre dois VPCs pode mudar a postura de segurança do. VPCs Além disso, adicionar novas entradas às tabelas de rotas pode alterar a postura de segurança dos recursos dentro do VPCs. É sua responsabilidade implementar essas configurações de forma a atender aos requisitos de segurança de sua organização. Para obter mais informações, consulte o Modelo de responsabilidade compartilhada.

O diagrama a seguir mostra os componentes do exemplo e as interações entre esses componentes.



Tópicos

- Pré-requisitos para usar o Amazon VPC no AWS Device Farm
- Etapa 1: Configurar uma conexão de emparelhamento entre VPC-1 e VPC-2
- Etapa 2: Atualizando as tabelas de rotas no VPC-1 e no VPC-2
- Etapa 3: Criar um grupo-alvo
- Etapa 4: Criando um Network Load Balancer
- Etapa 5: Criar um serviço de VPC endpoint para conectar sua VPC ao Device Farm
- Etapa 6: criar uma configuração de VPC endpoint entre sua VPC e o Device Farm
- Etapa 7: criar uma execução de teste para usar a configuração do VPC endpoint
- Criando uma rede escalável com o Transit Gateway

Pré-requisitos para usar o Amazon VPC no AWS Device Farm

Este exemplo requer o seguinte:

- Dois VPCs que são configurados com sub-redes contendo blocos CIDR não sobrepostos.
- A VPC-1 deve estar na região `us-west-2` e conter sub-redes para as zonas de disponibilidade `us-west-2a`, `us-west-2b` e `us-west-2c`.

Para obter mais informações sobre como criar VPCs e configurar sub-redes, consulte Trabalho com sub-redes VPCs e sub-redes no Amazon VPC Peering Guide.

Etapa 1: Configurar uma conexão de emparelhamento entre VPC-1 e VPC-2

Estabeleça uma conexão de emparelhamento entre os dois VPCs contendo blocos CIDR não sobrepostos. Para fazer isso, consulte Create and accept VPC peering connections no Guia de emparelhamento da Amazon VPC. Usando o cenário entre regiões deste tópico e o Guia de emparelhamento da Amazon VPC, é criado o seguinte exemplo de configuração de conexão de emparelhamento:

Nome

```
Device-Farm-Peering-Connection-1
```

ID VPC (solicitante)

```
vpc-0987654321gfedcba (VPC-2)
```

Conta

```
My account
```

Região

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

ID VPC (Aceitante)

```
vpc-1234567890abcdefg (VPC-1)
```

> **Note**
>
> Certifique-se de consultar as cotas de conexão de emparelhamento da VPC ao estabelecer novas conexões de emparelhamento. Para obter mais informações, consulte Amazon VPC quotas no Guia de emparelhamento da Amazon VPC.

Etapa 2: Atualizando as tabelas de rotas no VPC-1 e no VPC-2

Depois de configurar uma conexão de peering, você deve estabelecer uma rota de destino entre os dois VPCs para transferir dados entre eles. Para estabelecer essa rota, você pode atualizar manualmente a tabela de rotas da VPC-1 para apontar para a sub-rede da VPC-2 e vice-versa.

Para fazer isso, consulte Update your route tables for a VPC peering connection no Guia de emparelhamento da Amazon VPC. Usando o cenário entre regiões deste tópico e o Guia de emparelhamento da Amazon VPC, é criado o seguinte exemplo de configuração de tabela de rotas:

Exemplo de tabela de rotas VPC do Device Farm

Componente da VPC	VPC-1	VPC-2
ID da tabela de rotas	rtb-1234567890abcdefg	rtb-0987654321gfedcba
Faixa de endereços locais	10.0.0.0/16	172.16.0.0/16
Intervalo de endereços de destino	172.16.0.0/16	10.0.0.0/16

Etapa 3: Criar um grupo-alvo

Depois de definir as rotas de destino, você pode configurar um Network Load Balancer na VPC-1 para rotear as solicitações para a VPC-2.

O Network Load Balancer deve primeiro conter um grupo de destino que contenha os endereços IP para os quais as solicitações são enviadas.

Para criar um grupo de destino

1. Identifique os endereços IP do serviço que você deseja segmentar no VPC-2.
   - Esses endereços IP devem ser membros da sub-rede usada na conexão de emparelhamento.
   - Os endereços IP direcionados devem ser estáticos e imutáveis. Se o seu serviço tiver endereços IP dinâmicos, considere a possibilidade de direcionar um recurso estático (como um Network Load Balancer) e fazer com que esse recurso estático encaminhe as solicitações para o seu verdadeiro destino.

     > **Note**
     > - Se você tem como alvo uma ou mais instâncias autônomas do Amazon Elastic Compute Cloud (EC2Amazon), abra o console da EC2 Amazon https://console.aws.amazon.com/ec2/em e escolha Instâncias.
     > - Se você tem como alvo um grupo de instâncias da Amazon EC2 EC2 Auto Scaling, você deve associar o grupo Amazon Auto EC2 Scaling a um Network Load Balancer.

Para obter mais informações, consulte Como anexar um balanceador de carga ao seu grupo de Auto Scaling no Guia do usuário do Amazon Auto EC2 Scaling.

Em seguida, você pode abrir o EC2 console da Amazon em e https://console.aws.amazon.com/ec2/, em seguida, escolher Interfaces de rede. A partir daí, você pode visualizar os endereços IP de cada uma das interfaces de rede do Network Load Balancer em cada zona de disponibilidade.

2. Crie um grupo de destino na VPC-1. Para fazer isso, consulte Criar um grupo-alvo para o seu Network Load Balancer no Guia do Usuário para Network Load Balancers.

   Os grupos de destino para serviços em uma VPC diferente exigem a seguinte configuração:

   - Em Escolher um tipo de destino, escolha Endereços IP.
   - Para VPC, escolha a VPC que hospedará o balanceador de carga. Para o exemplo deste tópico, essa será a VPC-1.
   - Na página Registrar alvos, registre um alvo para cada endereço IP na VPC-2.

     Em Rede, escolha Outro endereço IP privado.

     Em Zona de disponibilidade, escolha as zonas desejadas na VPC-1.

     Para IPv4 endereço, escolha o endereço IP VPC-2.

     Em Portas, escolha suas portas.
   - Escolha Incluir como pendente abaixo. Quando terminar de especificar os endereços, selecione Registrar alvos pendentes.

Usando o cenário entre regiões deste tópico e o Guia do usuário de Network Load Balancers, os seguintes valores são usados na configuração do grupo de destino:

Target type

```
IP addresses
```

Nome do grupo-alvo

```
my-target-group
```

Protocolo/porta

```
TCP : 80
```

VPC

```
vpc-1234567890abcdefg (VPC-1)
```

Rede

```
Other private IP address
```

Zona de disponibilidade

```
all
```

IPv4 endereço

```
172.16.100.60
```

Portas

```
80
```

Etapa 4: Criando um Network Load Balancer

Crie um Network Load Balancer usando o grupo de destino descrito na etapa 3. Para fazer isso, consulte Criação de um Network Load Balancer.

Usando o cenário entre regiões deste tópico, os seguintes valores são usados em um exemplo de configuração do Network Load Balancer:

Nome do load balancer

```
my-nlb
```

Esquema

```
Internal
```

VPC

```
vpc-1234567890abcdefg (VPC-1)
```

Mapeamento

```
us-west-2a - subnet-4i23iuufkdiufsloi
```

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

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

Protocolo/porta

```
TCP : 80
```

Grupo de destino

```
my-target-group
```

Etapa 5: Criar um serviço de VPC endpoint para conectar sua VPC ao Device Farm

Você pode usar o Network Load Balancer para criar um serviço de endpoint da VPC. Por meio desse serviço de endpoint da VPC, o Device Farm pode se conectar ao seu serviço na VPC-2 sem nenhuma infraestrutura adicional, como um gateway da internet, uma instância NAT ou uma conexão VPN.

Para fazer isso, consulte Creating an Amazon VPC endpoint service.

Etapa 6: criar uma configuração de VPC endpoint entre sua VPC e o Device Farm

Agora você pode estabelecer uma conexão privada entre sua VPC e o Device Farm. Você pode usar o Device Farm para testar serviços privados sem expô-los à Internet pública. Para fazer isso, consulte Creating a VPC endpoint configuration in Device Farm.

Usando o cenário entre regiões deste tópico, os seguintes valores são usados em um exemplo de configuração de endpoint de VPC:

Nome

```
My VPCE Configuration
```

Nome do serviço VPCE

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

Nome DNS do serviço

```
devicefarm.com
```

Etapa 7: criar uma execução de teste para usar a configuração do VPC endpoint

Você pode criar execuções de teste que usam a configuração de endpoint da VPC descrita na etapa 6. Para ter mais informações, consulte Criando uma execução de teste no Device Farm ou Criar uma sessão.

Criando uma rede escalável com o Transit Gateway

Para criar uma rede escalável usando mais de duas VPCs, você pode usar o Transit Gateway para atuar como um hub de trânsito de rede para interconectar sua rede VPCs e a rede local. Para configurar uma VPC na mesma região que o Device Farm para usar um Transit Gateway, você pode seguir o guia Amazon VPC endpoint services with Device Farm para direcionar recursos em outra região com base em seus endereços IP privados.

Para obter mais informações sobre o Transit Gateway, consulte What is a transit gateway? no Guia de Transit Gateways da Amazon VPC.

Encerramento de dispositivos privados no Device Farm

`<Para encerrar um dispositivo privado após o prazo inicial acordado, você deve f`
Para obter mais informações sobre dispositivos privados, consulteDispositivos privados no AWS Device Farm.

> **Important**
>
> Essas instruções se aplicam somente à rescisão de contratos de dispositivos privados. Para todos os outros AWS serviços e problemas de cobrança, consulte a respectiva documentação desses produtos ou entre em contato com o AWS suporte.

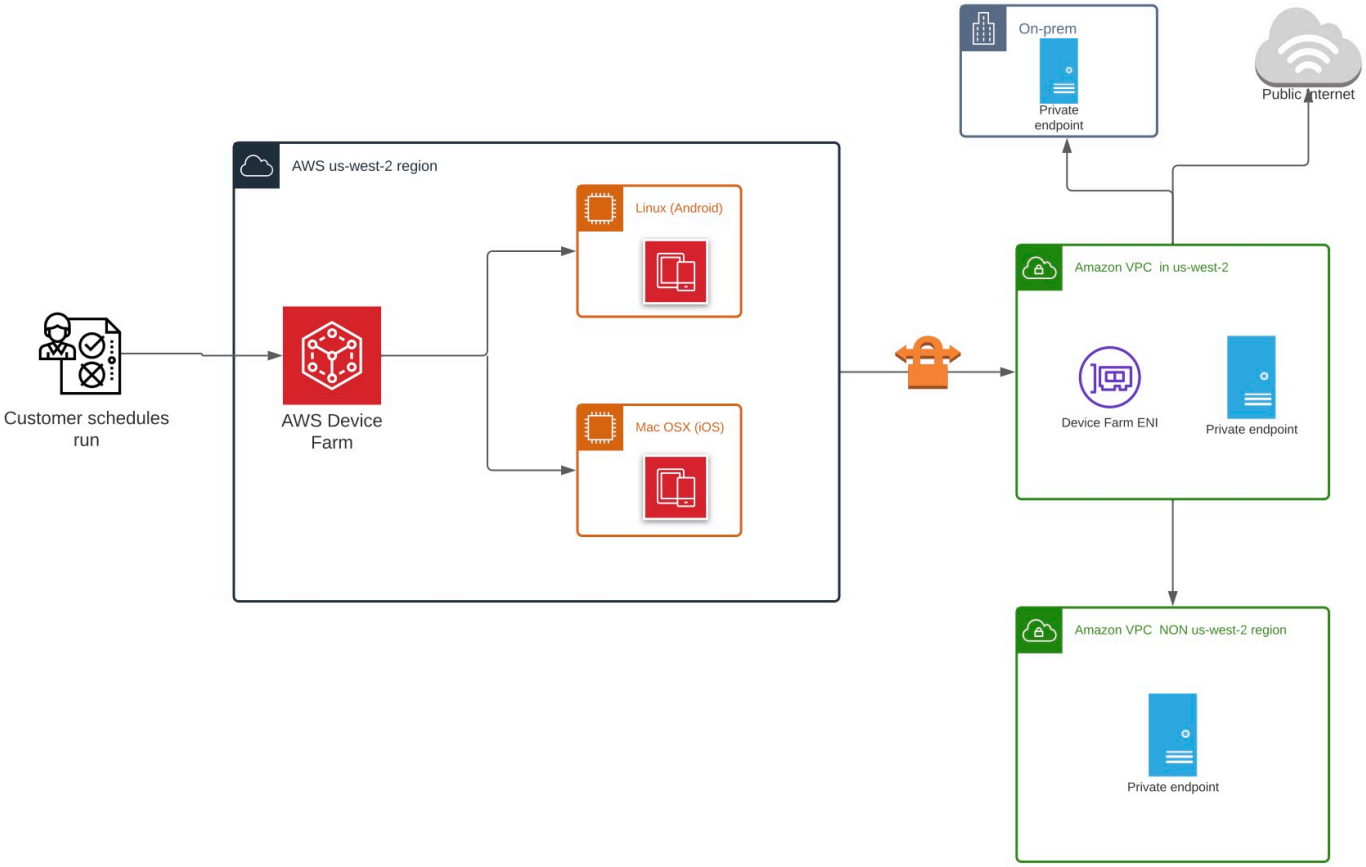
VPC-ENI no AWS Device Farm

> **Warning**
>
> Esse recurso só está disponível em dispositivos privados. Para solicitar o uso privado de um dispositivo em sua AWS conta, entre em contato conosco. Se você já tiver dispositivos privados adicionados à sua AWS conta, é altamente recomendável usar esse método de conectividade VPC.

O recurso de conectividade VPC-ENI do AWS Device Farm ajuda os clientes a se conectarem com segurança a seus endpoints privados hospedados no software local ou em outro provedor de AWS nuvem.

Você pode conectar os dispositivos móveis do Device Farm e suas máquinas host a um ambiente Amazon Virtual Private Cloud (Amazon VPC) na `us-west-2` região, o que permite o acesso a non-internet-facing serviços e aplicativos isolados por meio de uma interface de rede elástica. Para obter mais informações sobre VPCs, consulte o Guia do usuário da Amazon VPC.

Se a VPC ou o endpoint privado não estiver na região `us-west-2`, você poderá vinculá-lo a uma VPC na região `us-west-2` usando soluções como o Transit Gateway ou o emparelhamento da VPC. Nessas situações, o Device Farm criará uma ENI em uma sub-rede que você fornece para a VPC da região `us-west-2`, e você será responsável por garantir que uma conexão possa ser estabelecida entre a VPC da região `us-west-2` e a VPC na outra região.



Para obter informações sobre como usar AWS CloudFormation para criar e emparelhar automaticamente VPCs, consulte os VPCPeering modelos no repositório AWS CloudFormation de modelos em. GitHub

> **Note**
>
> O Device Farm não cobra nada pela criação ENIs na VPC de um cliente em. `us-west-2` O custo da conectividade entre regiões ou entre VPCs externas não está incluído nesse recurso.

Depois de configurar o acesso à VPC, os dispositivos e as máquinas host que você usa para seus testes não conseguirão se conectar a recursos fora da VPC (por exemplo, públicos CDNs), a menos

que haja um gateway NAT especificado dentro da VPC. Para obter mais informações, consulte Gateways NAT no Guia do usuário da Amazon VPC.

Tópicos

- AWS controle de acesso e IAM
- Perfis vinculados ao serviço
- Pré-requisitos
- Conectando o Amazon VPC
- Limites
- Usando serviços de endpoint da Amazon VPC com Device Farm - Legacy (não recomendado)

AWS controle de acesso e IAM

O AWS Device Farm permite que você use o AWS Identity and Access Management (IAM) para criar políticas que concedem ou restringem o acesso aos recursos do Device Farm. Para usar o recurso de conectividade VPC com o AWS Device Farm, a seguinte política do IAM é necessária para a conta de usuário ou função que você está usando para acessar o 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"
        }
      }
    }
  ]
}
```

Para criar ou atualizar um projeto Device Farm com uma configuração de VPC, sua política de IAM deve permitir que você chame as seguintes ações em relação aos recursos listados na configuração da VPC:

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

Além disso, sua política do IAM também deve permitir a criação do perfil vinculado ao serviço:

```
"iam:CreateServiceLinkedRole"
```

> **Note**
>
> Nenhuma dessas permissões é necessária para usuários que não usam configurações de VPC em seus projetos.

Perfis vinculados ao serviço

O AWS Device Farm usa AWS Identity and Access Management funções vinculadas a serviços (IAM). Um perfil vinculado ao serviço é um tipo exclusivo de perfil do IAM vinculado diretamente ao Device Farm. As funções vinculadas ao serviço são predefinidas pelo Device Farm e incluem todas as permissões que o serviço exige para chamar outros AWS serviços em seu nome.

Um perfil vinculado ao serviço facilita a configuração do Device Farm porque você não precisa adicionar manualmente as permissões necessárias. O Device Farm define as permissões de suas funções vinculadas ao serviço e, a menos que seja definido de outra forma, somente o Device Farm

pode assumir suas funções. As permissões definidas incluem a política de confiança e a política de permissões, e essa política não pode ser anexada a nenhuma outra entidade do IAM.

Uma função vinculada ao serviço poderá ser excluída somente após excluir seus recursos relacionados. Isso protege os recursos do Device Farm porque você não pode remover inadvertidamente a permissão para acessar os recursos.

Para obter informações sobre outros serviços que oferecem suporte aos perfis vinculados ao serviço, consulte Serviços da AWS que funcionam com o IAM e procure os serviços com Sim na coluna Perfil vinculado ao serviço. Escolha um Sim com um link para exibir a documentação da função vinculada a serviço desse serviço.

Permissões de perfil vinculado ao serviço para o Device Farm

O Device Farm usa a função vinculada ao serviço chamada AWSServiceRoleForDeviceFarm—Permite que o Device Farm acesse os recursos da AWS em seu nome.

A função AWSService RoleForDeviceFarm vinculada ao serviço confia nos seguintes serviços para assumir a função:

- `devicefarm.amazonaws.com`

A política de permissões de função permite que o Device Farm conclua as seguintes ações:

- Para sua conta
  - Criar interfaces de rede
  - Describe network interfaces
  - Descreva VPCs
  - Descrever sub-redes
  - Descrever grupos de segurança
  - Excluir interfaces
  - Modificar interfaces de rede
- Para interfaces de rede
  - Criar tags
- Para interfaces EC2 de rede gerenciadas pelo Device Farm
  - Criar permissões de interface de rede

A política do IAM completa diz:

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

Você deve configurar permissões para que uma entidade do IAM (por exemplo, um usuário, grupo ou função) crie, edite ou exclua um perfil vinculado a serviço. Para mais informações, consulte Permissões de perfil vinculado ao serviço no Guia do usuário do IAM.

Criar um perfil vinculado ao serviço para o Device Farm

Quando você fornece uma configuração de VPC para um projeto de teste móvel, você não precisa criar manualmente um perfil vinculado ao serviço. Quando você cria seu primeiro recurso do Device Farm na AWS Management Console, na ou na AWS API AWS CLI, o Device Farm cria a função vinculada ao serviço para você.

Se excluir esse perfil vinculado ao serviço e precisar criá-lo novamente, será possível usar esse mesmo processo para recriar o perfil em sua conta. Quando você cria seu primeiro recurso do Device Farm, o Device Farm cria o perfil vinculado ao serviço para você novamente.

Você também pode usar o console do IAM para criar um perfil vinculado ao serviço com o caso de uso do Device Farm. Na AWS CLI ou na AWS API, crie uma função vinculada ao serviço com o nome do `devicefarm.amazonaws.com` serviço. Para obter mais informações, consulte Criar uma função vinculada ao serviço no Guia do usuário do IAM. Se você excluir essa função vinculada ao serviço, será possível usar esse mesmo processo para criar a função novamente.

Editar um perfil vinculado ao serviço para o Device Farm

O Device Farm não permite que você edite a função AWSService RoleForDeviceFarm vinculada ao serviço. Depois que criar um perfil vinculado ao serviço, você não poderá alterar o nome do perfil, pois várias entidades podem fazer referência a ele. No entanto, será possível editar a descrição do perfil usando o IAM. Para obter mais informações, consulte Editar uma função vinculada a serviço no Guia do usuário do IAM.

Excluir um perfil vinculado ao serviço para o Device Farm

Se você não precisar mais usar um atributo ou serviço que requer uma função vinculada a serviço, é recomendável excluí-la. Dessa forma, você não tem uma entidade não utilizada que não seja monitorada ativamente ou mantida. No entanto, você deve limpar os recursos de seu perfil vinculado ao serviço antes de excluí-lo manualmente.

> **Note**
>
> Se o serviço Device Farm estiver usando a função quando você tentar excluir os recursos, a exclusão poderá falhar. Se isso acontecer, espere alguns minutos e tente a operação novamente.

Como excluir manualmente o perfil vinculado ao serviço usando o IAM

Use o console do IAM AWS CLI, o ou a AWS API para excluir a função AWSService RoleForDeviceFarm vinculada ao serviço. Para obter mais informações, consulte Excluir um perfil vinculado ao serviço no Guia do usuário do IAM.

Regiões compatíveis com as funções vinculadas ao serviço Device Farm

O Device Farm permite o uso de funções vinculadas a serviços em todas as regiões em que o serviço está disponível. Para mais informações, consulte Regiões e endpoints da AWS.

O Device Farm não permite o uso de funções vinculadas a serviços em todas as regiões em que o serviço está disponível. Você pode usar a AWSService RoleForDeviceFarm função nas seguintes regiões.

Nome da região	Identidade da região	Support em Device Farm
Leste dos EUA (Norte da Virgínia)	us-east-1	Não
Leste dos EUA (Ohio)	us-east-2	Não
Oeste dos EUA (N. da Califórnia)	us-west-1	Não
Oeste dos EUA (Oregon)	us-west-2	Sim
Ásia-Pacífico (Mumbai)	ap-south-1	Não
Ásia-Pacífico (Osaka)	ap-northeast-3	Não
Ásia-Pacífico (Seul)	ap-northeast-2	Não
Ásia-Pacífico (Singapura)	ap-southeast-1	Não

Nome da região	Identidade da região	Support em Device Farm
Ásia-Pacífico (Sydney)	ap-southeast-2	Não
Ásia-Pacífico (Tóquio)	ap-northeast-1	Não
Canadá (Central)	ca-central-1	Não
Europa (Frankfurt)	eu-central-1	Não
Europa (Irlanda)	eu-west-1	Não
Europa (Londres)	eu-west-2	Não
Europa (Paris)	eu-west-3	Não
América do Sul (São Paulo)	sa-east-1	Não
AWS GovCloud (US)	us-gov-west-1	Não

Pré-requisitos

A lista a seguir descreve alguns requisitos e sugestões a serem analisados ao criar configurações de VPC-ENI:

- Dispositivos privados devem ser atribuídos à sua AWS conta.
- Você deve ter uma AWS conta, usuário ou função com permissões para criar uma função vinculada ao serviço. Ao usar endpoints da Amazon VPC com os recursos de teste móvel do Device Farm, o Device Farm cria uma função vinculada ao AWS Identity and Access Management serviço (IAM).
- O Device Farm pode se conectar VPCs somente na `us-west-2` região. Se você não tiver uma VPC na região `us-west-2`, precisará criar uma. A seguir, para acessar recursos em uma VPC em outra região, você deve estabelecer uma conexão de emparelhamento entre a VPC na região `us-west-2` e a VPC na outra região. Para obter informações sobre peering VPCs, consulte o Amazon VPC Peering Guide.

Você deve verificar se tem acesso à sua VPC especificada ao configurar a conexão. Você deve configurar determinadas permissões do Amazon Elastic Compute Cloud (Amazon EC2) para Device Farm.

- A resolução de DNS é necessária na VPC que você usa.
- Depois que a VPC for criada, você precisará das seguintes informações sobre a VPC na região `us-west-2`:
  - ID da VPC
  - Sub-rede IDs (somente sub-redes privadas)
  - Grupo de segurança IDs
- Você deve configurar as conexões da Amazon VPC por projeto. No momento, você pode configurar somente uma configuração de VPC por projeto. Quando você configura uma VPC, a Amazon VPC cria uma interface dentro da sua VPC e a atribui às sub-redes e grupos de segurança especificados. Todas as sessões futuras associadas ao projeto usarão a conexão VPC configurada.
- Você não pode usar as configurações do VPC-ENI junto com o recurso VPCE legado.
- É altamente recomendável não atualizar um projeto existente com uma configuração de VPC-ENI, pois os projetos existentes podem ter configurações de VPCE que persistem no nível de execução. Em vez disso, se você já usa os recursos existentes do VPCE, use o VPC-ENI para todos os novos projetos.

Conectando o Amazon VPC

Você pode configurar e atualizar seu projeto para usar endpoints da Amazon VPC. A configuração da VPC-ENI é configurada para cada projeto. Um projeto só pode ter um endpoint da VPC-ENI por vez. Para configurar o acesso da VPC a um projeto, você deve conhecer os seguintes detalhes:

- O ID da VPC em `us-west-2` se a aplicação estiver hospedada lá, ou o ID da VPC de `us-west-2` que se conecta a outra VPC em uma região diferente.
- Os grupos de segurança aplicáveis a serem aplicados à conexão.
- As sub-redes que serão associadas à conexão. Quando uma sessão é iniciada, a maior sub-rede disponível é usada. Recomendamos ter várias sub-redes associadas a diferentes zonas de disponibilidade para melhorar a postura de disponibilidade da sua conectividade de VPC.

Depois de criar sua configuração de VPC-ENI, você pode atualizar seus detalhes usando o console ou a CLI usando as etapas abaixo.

Console

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e, em seguida, Projetos.
3. Em Projetos de teste móvel, escolha o nome do seu projeto na lista.
4. Escolha Configurações do projeto.
5. Na seção Configurações da Virtual Private Cloud (VPC), você pode alterar a`VPC`, `Subnets` (somente sub-redes privadas) e. `Security Groups`
6. Escolha Salvar.

CLI

Use o seguinte comando AWS CLI para atualizar o 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
```

Você também pode configurar uma Amazon VPC ao criar seu projeto:

```
$  aws devicefarm create-project \
--name VPCDemo \
--vpc-config \
securityGroupIds=sg-02c1537701a7e3763,sg-005dadf9311efda25,\
subnetIds=subnet-09b1a45f9cac53717,subnet-09b1a45f9cac12345,\
vpcId=vpc-0238fb322af81a368
```

Limites

As seguintes limitações são aplicáveis ao recurso VPC-ENI:

- Você pode fornecer até cinco grupos de segurança na configuração de VPC de um projeto Device Farm.
- Você pode fornecer até oito sub-redes na configuração de VPC de um projeto Device Farm.
- Ao configurar um projeto do Device Farm para funcionar com sua VPC, a menor sub-rede que você pode fornecer deve ter no mínimo cinco endereços disponíveis. IPv4
- Não há suporte para endereços IP públicos no momento. Em vez disso, recomendamos usar sub-redes privadas em seus projetos do Device Farm. Se precisar de acesso público à Internet durante os testes, use um gateway de conversão de endereços de rede (NAT). Configurar um projeto do Device Farm com uma sub-rede pública não dá a seus testes acesso à Internet nem a um endereço IP público.
- A integração VPC-ENI só oferece suporte a sub-redes privadas em sua VPC.
- Somente o tráfego de saída da ENI gerenciada pelo serviço é permitido. Isso significa que a ENI não pode receber solicitações de entrada não solicitadas da VPC.

Usando serviços de endpoint da Amazon VPC com Device Farm - Legacy (não recomendado)

Warning

É altamente recomendável usar a conectividade VPC-ENI descrita nesta página para conectividade de endpoints privados, pois o VPCE agora é considerado um recurso legado. O VPC-ENI oferece mais flexibilidade, configurações mais simples, é mais econômico e requer significativamente menos sobrecarga de manutenção quando comparado ao método de conectividade VPCE.

Note

O uso de serviços de endpoint da VPC da Amazon com o Device Farm só é permitido para clientes com dispositivos privados configurados. Para habilitar sua conta AWS para usar esse recurso com dispositivos privados, entre em contato conosco.

A Amazon Virtual Private Cloud (Amazon VPC) é um AWS serviço que você pode usar para lançar AWS recursos em uma rede virtual que você define. Com uma VPC, você tem controle sobre suas

configurações de rede, como o intervalo de endereços IP, sub-redes, tabelas de roteamento e gateways de rede.

Se você usa a Amazon VPC para hospedar aplicativos privados na AWS região Oeste dos EUA (Oregon) (`us-west-2`), você pode estabelecer uma conexão privada entre sua VPC e o Device Farm. Com essa conexão, você pode usar o Device Farm para testar aplicações privadas sem expô-las à Internet pública. Para permitir que sua AWS conta use esse recurso com dispositivos privados, entre em contato conosco.

Para conectar um recurso em na VPC ao Device Farm, você pode usar o console da Amazon VPC para criar um serviço de endpoint da VPC. Esse serviço de endpoint permite que você forneça o recurso na VPC ao Device Farm por meio de um endpoint da VPC do Device Farm. O serviço de endpoint fornece conectividade confiável e escalável ao Device Farm sem exigir um gateway de Internet, uma instância de conversão de endereços de rede (NAT) ou uma conexão VPN. Para obter mais informações, consulte VPC endpoint services (AWS PrivateLink) no Guia.AWS PrivateLink

> **Important**
>
> O recurso de endpoint VPC do Device Farm ajuda você a conectar com segurança serviços internos privados em sua VPC à VPC pública do Device Farm usando conexões. AWS PrivateLink Embora a conexão seja segura e privada, essa segurança depende da proteção de suas credenciais da AWS . Se suas AWS credenciais forem comprometidas, um invasor poderá acessar ou expor seus dados de serviço para o mundo externo.

Depois de criar um serviço de endpoint da VPC na Amazon VPC, você pode usar o console do Device Farm para criar uma configuração de endpoint da VPC no Device Farm. Este tópico mostra como criar a conexão da Amazon VPC e a configuração do endpoint da VPC no Device Farm.

Antes de começar

As informações a seguir são para usuários da Amazon VPC na região oeste dos EUA (Oregon) (`us-west-2`), com uma sub-rede em cada uma das seguintes zonas de disponibilidade: us-west-2a, us-west-2b e us-west-2c.

O Device Farm tem requisitos adicionais para os serviços de endpoint da VPC com os quais você pode usá-lo. Quando você criar e configurar um serviço de endpoint da VPC para trabalhar com o Device Farm, escolha opções que atendam aos seguintes requisitos:

- As zonas de disponibilidade do serviço devem incluir us-west-2a, us-west-2b e us-west-2c. O Network Load Balancer associado a um serviço de endpoint da VPC determina as zonas de disponibilidade para esse serviço de endpoint da VPC. Se o serviço de endpoint da VPC não mostrar todas as três zonas de disponibilidade, você deverá recriar o Network Load Balancer para ativar essas três zonas e, em seguida, associar novamente o Network Load Balancer ao serviço de endpoint.
- As entidades principais permitidas para o serviço de endpoint devem incluir o nome do recurso da Amazon (ARN) do endpoint da VPC do Device Farm (ARN do serviço). Depois de criar o serviço de endpoint, adicione o ARN do serviço de endpoint da VPC do Device Farm à sua lista de permissões para dar permissão ao Device Farm para acessar seu serviço de endpoint da VPC. Para obter o ARN do serviço de endpoint da VPC do Device Farm, fale conosco.

Além disso, se mantiver a configuração Aceitação necessária ativada ao criar o serviço de endpoint da VPC, você deverá aceitar manualmente cada solicitação de conexão que o Device Farm enviar ao serviço de endpoint. Para alterar essa configuração de um serviço de endpoint existente, escolha o serviço de endpoint no console da Amazon VPC, selecione Ações e Modificar configuração de aceitação do endpoint. Para obter mais informações, consulte Alterar os balanceadores de carga e as configurações de aceitação no Guia do AWS PrivateLink .

A próxima seção explica como criar um serviço de endpoint da Amazon VPC que atenda a esses requisitos.

Etapa 1: Criação de um Network Load Balancer

A primeira etapa para estabelecer uma conexão privada entre sua VPC e o Device Farm é criar um Network Load Balancer para rotear as solicitações para um grupo de destino.

New console

Para criar um Network Load Balancer usando o novo console

1. Abra o console do Amazon Elastic Compute Cloud (Amazon EC2) em https://console.aws.amazon.com/ec2/.
2. No painel de navegação, em Balanceamento de carga, escolha Balanceadores de carga.
3. Selecione Criar um balanceador de carga.
4. Em Network Load Balancer, escolha Criar.
5. Na página Criar Network Load Balancer, em Configuração básica, faça o seguinte:

a. Insira um Nome para o balanceador de carga.

b. Em Esquema, escolha Interno.

6. Em Mapeamento de rede, faça o seguinte:

a. Escolha a VPC do grupo de destino.

b. Selecione os seguintes Mapeamentos:

- `us-west-2a`
- `us-west-2b`
- `us-west-2c`

7. Em Receptores e roteamento, use as opções Protocolo e Porta para escolher o grupo de destino.

> **Note**
>
> Por padrão, o balanceamento de carga da zona de disponibilidade cruzada está desativado.
> Como o balanceador de carga usa as zonas de disponibilidade `us-west-2a`, `us-west-2b` e `us-west-2c`, ele exige que os destinos sejam registrados em cada uma delas ou, se você registrar destinos em menos de todas as três zonas, exige a ativação do balanceamento de carga entre zonas. Caso contrário, o balanceador de carga pode não funcionar como esperado.

8. Selecione Criar um balanceador de carga.

Old console

Para criar um Network Load Balancer usando o console antigo

1. Abra o console do Amazon Elastic Compute Cloud (Amazon EC2) em https://console.aws.amazon.com/ec2/.
2. No painel de navegação, em Balanceamento de carga, escolha Balanceadores de carga.
3. Selecione Criar um balanceador de carga.
4. Em Network Load Balancer, escolha Criar.
5. Na página Configurar balanceador de carga, em Configuração básica, faça o seguinte:

a. Insira um Nome para o balanceador de carga.

b. Em Esquema, escolha Interno.

6. Em Receptores, selecione o Protocolo e a Porta que o grupo de destino está usando.

7. Em Zonas de disponibilidade, faça o seguinte:

a. Escolha a VPC do grupo de destino.

b. Selecione as seguintes zonas de disponibilidade:

- `us-west-2a`
- `us-west-2b`
- `us-west-2c`

c. Escolha Próximo: defina as configurações de segurança.

8. (Opcional) Defina suas configurações de segurança e, em seguida, selecione Próximo: configure o roteamento.

9. Na página Configurar roteamento, faça o seguinte:

a. Em Grupo-alvo, selecione Grupo-alvo existente.

b. Para Nome, escolha seu grupo-alvo.

c. Escolha Próximo: registrar alvos.

10. Na página Registrar alvos, revise seus alvos e escolha Próximo: revisão.

> **Note**
>
> Por padrão, o balanceamento de carga da zona de disponibilidade cruzada está desativado.
> Como o balanceador de carga usa as zonas de disponibilidade `us-west-2a`, `us-west-2b` e `us-west-2c`, ele exige que os destinos sejam registrados em cada uma delas ou, se você registrar destinos em menos de todas as três zonas, exige a ativação do balanceamento de carga entre zonas. Caso contrário, o balanceador de carga pode não funcionar como esperado.

11. Revise a configuração do balanceador de carga e escolha Criar.

Etapa 2: criar um serviço de endpoint da Amazon VPC

Depois de criar o Network Load Balancer, use o console da Amazon VPC para criar um serviço de endpoint na VPC.

1. Abra o console da Amazon VPC em https://console.aws.amazon.com/vpc/.
2. Em Recursos por região, escolha Serviços de endpoint.
3. Escolha Criar serviço de endpoint.
4. Execute um destes procedimentos:

   - Se você já tiver um Network Load Balancer que deseja que o serviço de endpoint use, escolha-o em Balanceadores de carga disponíveis e passe para a etapa 5.
   - Se você ainda não tiver criado um Network Load Balancer, escolha Criar novo balanceador de carga. O EC2 console da Amazon é aberto. Siga as etapas em Criar um Network Load Balancer, começando pela etapa 3, e continue com essas etapas no console da Amazon VPC.

5. Para as zonas de disponibilidade incluídas, verifique se `us-west-2a`, `us-west-2b` e `us-west-2c` aparecem na lista.
6. Se você não quiser aceitar ou negar manualmente cada solicitação de conexão enviada ao serviço de endpoint, em Configurações adicionais, desmarque Aceitação necessária. Se você desmarcar essa caixa de seleção, o serviço endpoint aceitará automaticamente cada solicitação de conexão que receber.
7. Escolha Criar.
8. No novo serviço de endpoint, escolha Permitir principais.
9. Fale conosco para obter o ARN do endpoint da VPC do Device Farm (ARN do serviço) a ser adicionado à lista de permissões do serviço de endpoint e, depois, adicione esse ARN do serviço à lista de permissões do serviço.
10. Na guia Detalhes do serviço endpoint, anote o nome do serviço - (nome do serviço). Você precisará desse nome ao criar a configuração do VPC endpoint na próxima etapa.

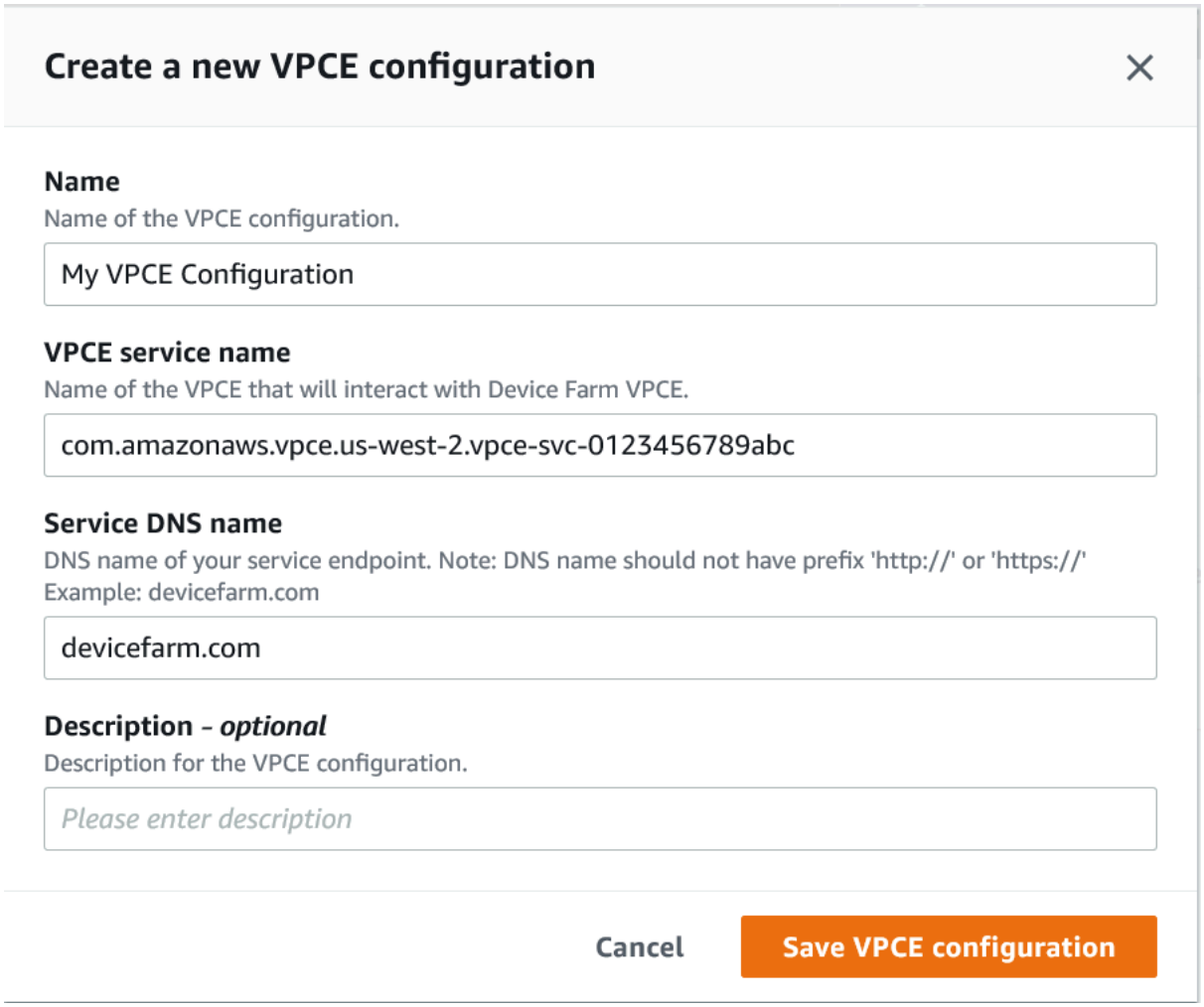
O serviço de endpoint da VPC agora está pronto para ser usado com o Device Farm.

Etapa 3: criar uma configuração de endpoint da VPC no Device Farm

Depois de criar um serviço de endpoint na Amazon VPC, você pode criar uma configuração de endpoint da Amazon VPC no Device Farm.

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. No painel de navegação, escolha Teste para dispositivos móveis e Dispositivos privados.
3. Escolha Configurações de VPCE.
4. Escolha Criar uma configuração de VPCE.
5. Em Criar uma nova configuração de VPCE, insira um Nome para a configuração do endpoint da VPC.
6. Em Nome do serviço de VPCE, insira o nome do serviço de endpoint da Amazon VPC (nome do serviço) que você anotou no console da Amazon VPC. O nome é semelhante ao `com.amazonaws.vpce.us-west-2.vpce-svc-id`.
7. Em Nome DNS do serviço, digite o nome DNS do serviço para a aplicação que você deseja testar (por exemplo, `devicefarm.com`). Não especifique `http` ou `https` antes do nome DNS do serviço.

   O nome de domínio não é acessível pela internet pública. Além disso, esse novo nome de domínio, que mapeia para o serviço de endpoint da VPC, é gerado pelo Amazon Route 53 e está disponível exclusivamente para você na sua sessão do Device Farm.
8. Escolha Salvar.



Etapa 4: Criar uma execução de teste

Depois de salvar a configuração do endpoint da VPC, você pode usar a configuração para criar execuções de teste ou sessões de acesso remoto. Para ter mais informações, consulte Criando uma execução de teste no Device Farm ou Criar uma sessão.

Registro de chamadas de API do AWS Device Farm com AWS CloudTrail

O AWS Device Farm é integrado com AWS CloudTrail, um serviço que fornece um registro das ações realizadas por um usuário, função ou AWS serviço no AWS Device Farm. CloudTrail captura todas as chamadas de API para o AWS Device Farm como eventos. As chamadas capturadas incluem chamadas do console do AWS Device Farm e chamadas de código para as operações da API do AWS Device Farm. Se você criar uma trilha, poderá habilitar a entrega contínua de CloudTrail eventos para um bucket do Amazon S3, incluindo eventos para o AWS Device Farm. Se você não configurar uma trilha, ainda poderá ver os eventos mais recentes no CloudTrail console no Histórico de eventos. Usando as informações coletadas por CloudTrail, você pode determinar a solicitação que foi feita ao AWS Device Farm, o endereço IP a partir do qual a solicitação foi feita, quem fez a solicitação, quando ela foi feita e detalhes adicionais.

Para saber mais sobre isso CloudTrail, consulte o Guia AWS CloudTrail do usuário.

Informações do AWS Device Farm em CloudTrail

CloudTrail é ativado em sua AWS conta quando você cria a conta. Quando a atividade ocorre no AWS Device Farm, essa atividade é registrada em um CloudTrail evento junto com outros eventos AWS de serviço no histórico de eventos. Você pode visualizar, pesquisar e baixar eventos recentes em sua AWS conta. Para obter mais informações, consulte Visualização de eventos com histórico de CloudTrail eventos.

Para um registro contínuo dos eventos em sua AWS conta, incluindo eventos do AWS Device Farm, crie uma trilha. Uma trilha permite CloudTrail entregar arquivos de log para um bucket do Amazon S3. Por padrão, quando você cria uma trilha no console, ela é aplicada a todas as regiões da AWS. A trilha registra eventos de todas as regiões na AWS partição e entrega os arquivos de log ao bucket do Amazon S3 que você especificar. Além disso, você pode configurar outros AWS serviços para analisar e agir com base nos dados de eventos coletados nos CloudTrail registros. Para obter mais informações, consulte:

- Visão Geral para Criar uma Trilha
- CloudTrail Serviços e integrações compatíveis
- Configurando notificações do Amazon SNS para CloudTrail

- Recebendo arquivos de CloudTrail log de várias regiões e recebendo arquivos de CloudTrail log de várias contas

Quando o CloudTrail registro está ativado em sua AWS conta, as chamadas de API feitas para as ações do Device Farm são rastreadas em arquivos de log. Os registros do Device Farm são gravados junto com outros registros de AWS serviço em um arquivo de log. CloudTrail determina quando criar e gravar em um novo arquivo com base no período e no tamanho do arquivo.

Todas as ações do Device Farm são registradas em log e documentadas em AWS CLI referência e Automatização do Device Farm. Por exemplo, chamadas para criar um novo projeto ou executar no Device Farm geram entradas em arquivos de CloudTrail log.

Cada entrada de log ou evento contém informações sobre quem gerou a solicitação. As informações de identidade ajudam a determinar o seguinte:

- Se a solicitação foi feita com credenciais de usuário root ou AWS Identity and Access Management (IAM).
- Se a solicitação foi feita com credenciais de segurança temporárias de uma função ou de um usuário federado.
- Se a solicitação foi feita por outro AWS serviço.

Para obter mais informações, consulte Elemento userIdentity do CloudTrail .

Compreensão das entradas do arquivo de log do AWS Device Farm

Uma trilha é uma configuração que permite a entrega de eventos como arquivos de log para um bucket do Amazon S3 que você especificar. CloudTrail os arquivos de log contêm uma ou mais entradas de log. Um evento representa uma única solicitação de qualquer fonte e inclui informações sobre a ação solicitada, a data e a hora da ação, os parâmetros da solicitação e assim por diante. CloudTrail os arquivos de log não são um rastreamento de pilha ordenado das chamadas públicas de API, portanto, eles não aparecem em nenhuma ordem específica.

O exemplo a seguir mostra uma entrada de CloudTrail registro que demonstra a `ListRuns` ação Device Farm:

```
{
```

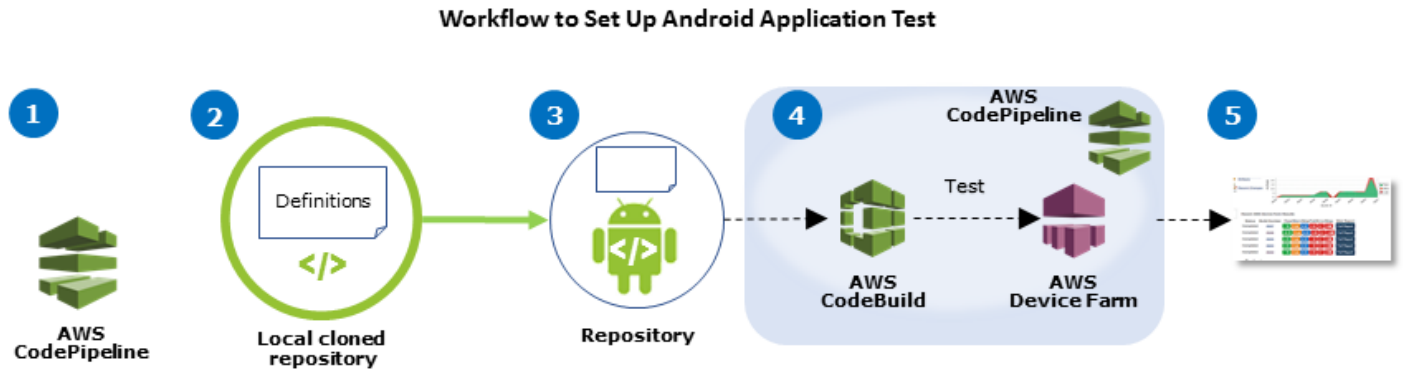
```
  "Records": [
    {
      "eventVersion": "1.03",
      "userIdentity": {
        "type": "Root",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:root",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2015-07-08T21:13:35Z"
          }
        }
      },
      "eventTime":"2015-07-09T00:51:22Z",
      "eventSource": "devicefarm.amazonaws.com",
      "eventName":"ListRuns",
      "awsRegion":"us-west-2",
      "sourceIPAddress":"203.0.113.11",
      "userAgent":"example-user-agent-string",
      "requestParameters": {
        "arn":"arn:aws:devicefarm:us-west-2:123456789012:project:a9129b8c-
df6b-4cdd-8009-40a25EXAMPLE"},
        "responseElements": {
          "runs": [
            {
              "created": "Jul 8, 2015 11:26:12 PM",
              "name": "example.apk",
              "completedJobs": 2,
              "arn": "arn:aws:devicefarm:us-west-2:123456789012:run:a9129b8c-
df6b-4cdd-8009-40a256aEXAMPLE/1452d105-e354-4e53-99d8-6c993EXAMPLE",
              "counters": {
                "stopped": 0,
                "warned": 0,
                "failed": 0,
                "passed": 4,
                "skipped": 0,
                "total": 4,
                "errored": 0
              },
              "type": "BUILTIN_FUZZ",
              "status": "RUNNING",
```

```
              "totalJobs": 3,
              "platform": "ANDROID_APP",
              "result": "PENDING"
            },
            ... additional entries ...
          ]
        }
      }
    }
  ]
}
```

Integração do AWS Device Farm em um estágio de CodePipeline teste

Você pode usar o AWS CodePipeline para incorporar testes de aplicações móveis configuradas no Device Farm em um pipeline de lançamento automatizado gerenciado pela AWS. Você pode configurar o pipeline para executar testes sob demanda, em uma programação, ou como parte de um fluxo de integração contínua.

O diagrama a seguir mostra o fluxo de integração contínua em que um aplicativo Android é criado e testado cada vez que um envio é confirmado para o repositório. Para criar essa configuração de pipeline, consulte o tutorial: Crie e teste um aplicativo Android quando enviado para GitHub.



1. Configurar	2. Adicionar definições	3. Push	4. Criar e testar	5. Relatório
Configurar recursos do pipeline	Adicionar definições de criação e teste ao seu pacote	Enviar um pacote para o seu repositório	Compilação e teste de aplicativo de artefato de saída de compilação iniciada automaticamente	Visualizar resultados do teste

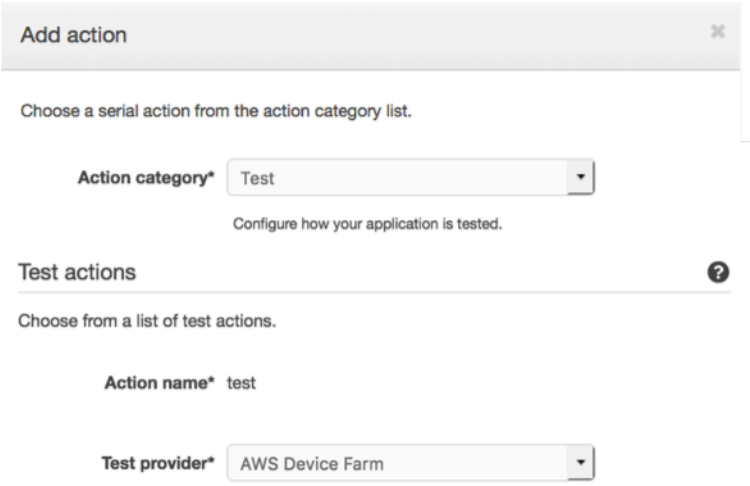
Para saber como configurar um pipeline que testa continuamente um aplicativo compilado (como um arquivo iOS `.ipa` ou arquivo Android `.apk`) como sua origem, consulte Tutorial: Testar um aplicativo iOS toda vez que você carregar um arquivo .ipa em um bucket do Amazon S3.

Configure CodePipeline para usar seus testes do Device Farm

Nessas etapas, presumimos que você tenha configurado um projeto do Device Farm e criado um pipeline. O pipeline deve ser configurado com um estágio de teste que recebe um artefato de entrada que contém a definição do teste e os arquivos do pacote do aplicativo compilado. O artefato de entrada do estágio de teste pode ser o artefato de saída de um estágio de origem ou de compilação configurado no pipeline.

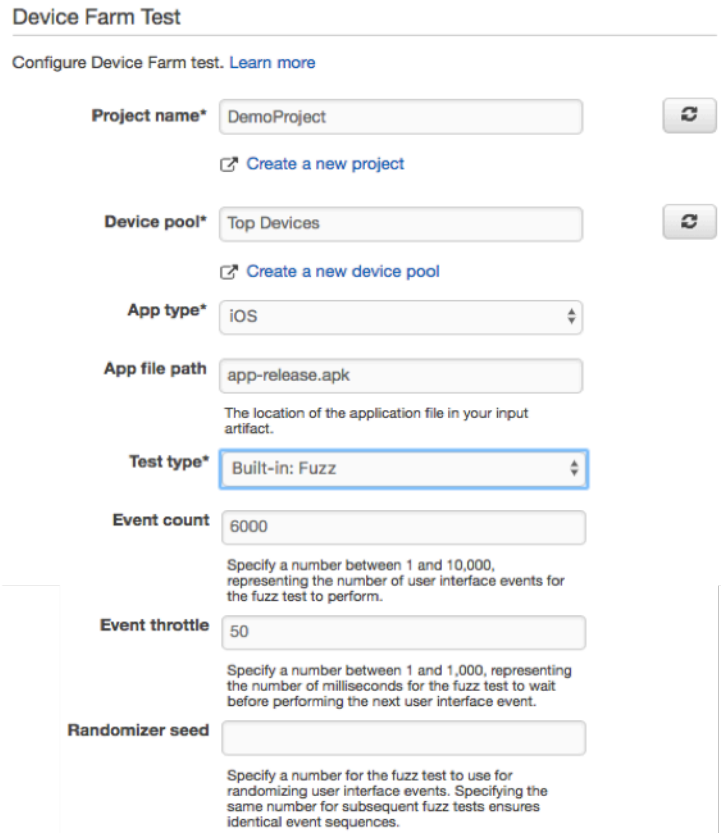
Para configurar um teste do Device Farm, execute como uma ação CodePipeline de teste

1. Faça login no AWS Management Console e abra o CodePipeline console em https://console.aws.amazon.com/codepipeline/.
2. Selecione o pipeline para a versão de seu aplicativo.
3. No painel do estágio de teste, selecione o ícone de lápis e, em seguida, selecione Ação.
4. No painel Adicionar ação, em Categoria da ação, escolha Teste.
5. Em Nome da ação, insira um nome.
6. Em Provedor do teste, selecione AWS Device Farm.



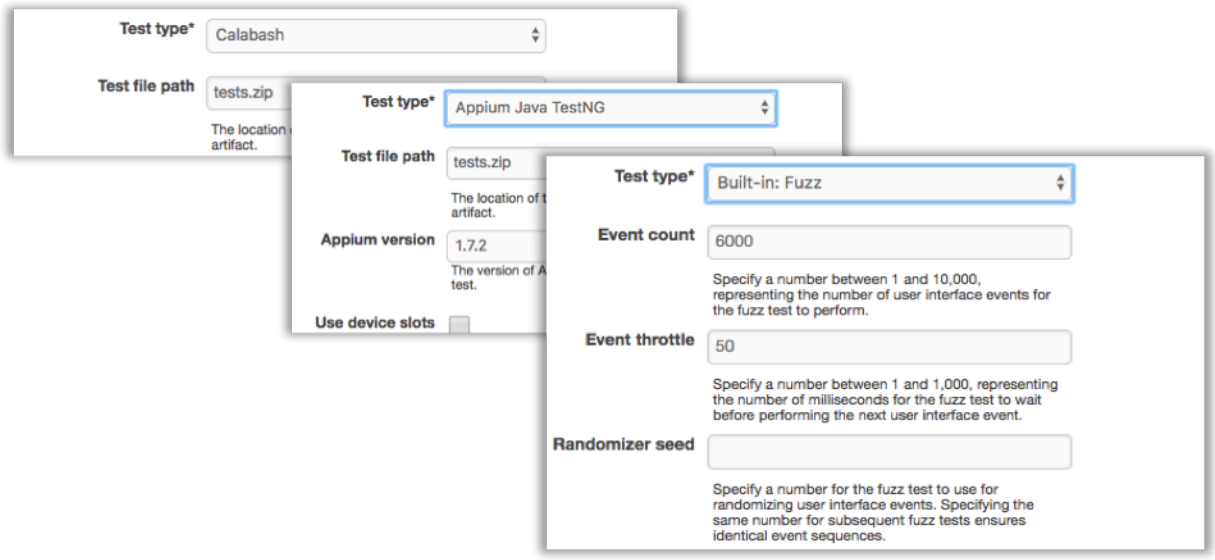
7. Em Nome do projeto, selecione o projeto existente do Device Farm ou selecione Criar um novo projeto.

8. Em Grupo de dispositivos, selecione seu grupo de dispositivos existente ou selecione Criar um novo grupo de dispositivos. Se você criar um grupo de dispositivos, será necessário selecionar um conjunto de dispositivos de teste.
9. Em Tipo de aplicação, selecione a plataforma para sua aplicação.

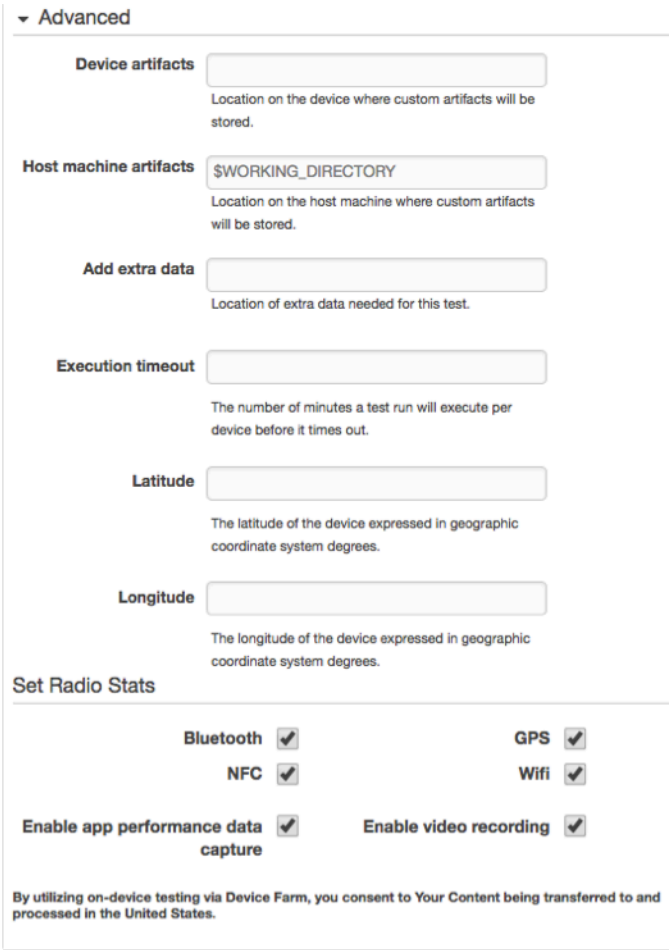


10. Em Caminho para o arquivo da aplicação, insira o caminho do pacote da aplicação compilada. O caminho é relativo à raiz do artefato de entrada para o teste.
11. Em Tipo de teste, siga um destes procedimentos:

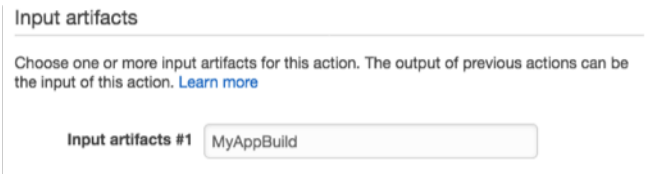
    - Se estiver usando um dos testes internos do Device Farm, escolha o tipo de teste configurado no projeto do Device Farm.
    - Se não estiver usando um dos testes incorporados do Device Farm, no Caminho do arquivo de teste, insira o caminho do arquivo de definição do teste. O caminho é relativo à raiz do artefato de entrada para o teste.



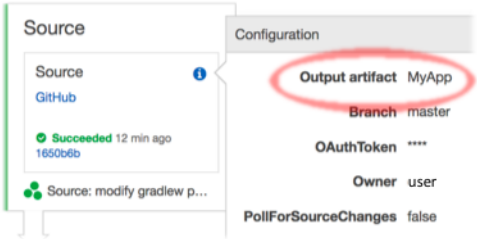
12. Nos campos restantes, forneça a configuração que seja apropriada para seu teste e tipo de aplicativo.
13. (Opcional) Em Avançado, forneça uma configuração detalhada para a execução do teste.



14. Em Artefatos de entrada, selecione o artefato de entrada que corresponde ao artefato de saída do estágio que vem antes do estágio de teste no pipeline.



No CodePipeline console, você pode encontrar o nome do artefato de saída para cada estágio passando o mouse sobre o ícone de informações no diagrama do pipeline. Se seu pipeline testar seu aplicativo diretamente do estágio de origem, escolha MyApp. Se o seu funil incluir um estágio de construção, escolha MyAppBuild.



15. Na parte inferior do painel, selecione Adicionar ação.
16. No CodePipeline painel, escolha Salvar alteração do pipeline e escolha Salvar alteração.
17. Para enviar suas alterações e iniciar uma compilação do pipeline, selecione Liberar alteração e, depois, Liberar.

AWS CLI referência para AWS Device Farm

Para usar o AWS Command Line Interface (AWS CLI) para executar comandos do Device Farm, consulte a AWS CLI referência do AWS Device Farm.

Para obter informações gerais sobre o AWS CLI, consulte o Guia AWS Command Line Interface do Usuário e a Referência de AWS CLI Comandos.

PowerShell Referência do Windows para o AWS Device Farm

Para usar o Windows PowerShell para executar comandos do Device Farm, consulte a Referência do Cmdlet do Device Farm na Referência do AWS Tools for Windows PowerShell Cmdlet. Para obter mais informações, consulte Como configurar as ferramentas da AWS para Windows PowerShell no Guia Ferramentas da AWS para PowerShell do usuário.

Automatização do AWS Device Farm

O acesso programático ao Device Farm é uma maneira eficiente de automatizar as tarefas comuns que você precisa realizar, como agendar uma execução ou fazer download dos artefatos de uma execução, suíte ou teste. O AWS SDK e AWS CLI forneça meios para fazer isso.

O AWS SDK fornece acesso a todos os AWS serviços, incluindo Device Farm, Amazon S3 e muito mais. Para obter mais informações, consulte .

- as AWS ferramentas e SDKs
- Referência da API do AWS Device Farm

Exemplo: usar o AWS SDK para iniciar uma execução do Device Farm e coletar artefatos

O exemplo a seguir beginning-to-end mostra como você pode usar o AWS SDK para trabalhar com o Device Farm. Esse exemplo faz o seguinte:

- Carrega um teste e pacotes de aplicações para o Device Farm
- Inicia uma execução de teste e aguarda sua conclusão (ou falha)
- Faz download de todos os artefatos produzidos pelos conjuntos de testes

Esse exemplo depende do pacote `requests` de terceiros para interagir com 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.asc

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")
```

Solução de problemas de Device Farm

Nesta seção, você encontrará mensagens de erro e procedimentos para ajudá-lo a corrigir problemas comuns com o Device Farm.

Tópicos

- Solução de problemas de testes de aplicações para Android no AWS Device Farm
- Solução de problemas de JUnit testes do Appium Java no AWS Device Farm
- Solução de problemas de testes de aplicativos JUnit web Appium Java no AWS Device Farm
- Solução de problemas de testes Appium Java TestNG no AWS Device Farm
- Solução de problemas de aplicações Web Appium Java TestNG no AWS Device Farm
- Solução de problemas de testes Appium Python no AWS Device Farm
- Solução de problemas de testes de aplicações Web Appium Python no AWS Device Farm
- Solução de problemas de testes de instrumentação no AWS Device Farm
- Solução de problemas de testes de aplicações iOS no AWS Device Farm
- XCTest Testes de solução de problemas no AWS Device Farm
- Solução de problemas de testes de XCTest interface do usuário no AWS Device Farm

Solução de problemas de testes de aplicações para Android no AWS Device Farm

O tópico a seguir lista mensagens de erro que ocorrem durante o upload de testes de aplicativos Android e recomenda soluções para resolver cada erro.

> **Note**
>
> As instruções a seguir baseiam-se no Linux x86_64 e Mac.

ANDROID_APP_UNZIP_FAILED

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não conseguimos abrir seu aplicativo. Verifique se o arquivo é válido e tente novamente.

Verifique se você consegue descompactar o pacote de aplicativos sem erros. No exemplo a seguir, o nome do pacote é app-debug.apk.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip app-debug.apk
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Um pacote de aplicativos Android válido deve gerar um resultado semelhante ao seguinte:

   ```
   .
   |-- AndroidManifest.xml
   |-- classes.dex
   |-- resources.arsc
   |-- assets (directory)
   |-- res (directory)
   `-- META-INF (directory)
   ```

   Para obter mais informações, consulte Testes de Android no AWS Device Farm.

ANDROID_APP_AAPT_DEBUG_BADGING_FAILED

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível extrair informações sobre seu aplicativo. Para verificar se o aplicativo é válido, execute o comando `aapt debug badging` *`<path to your test package>`* e tente novamente se o comando não imprimir nenhum erro.

Durante o processo de validação de upload, o AWS Device Farm analisa as informações da saída de um comando `aapt debug badging` *<path to your package>*.

Verifique se você consegue executar esse comando com êxito em seu aplicativo Android. No exemplo a seguir, o nome do pacote é app-debug.apk.

- Copie o pacote do aplicativo para seu diretório de trabalho e, em seguida, execute o comando:

```
$ aapt debug badging app-debug.apk
```

  Um pacote de aplicativos Android válido deve gerar um resultado semelhante ao seguinte:

```
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-implied-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'
```

  Para obter mais informações, consulte Testes de Android no AWS Device Farm.

ANDROID_APP_PACKAGE_NAME_VALUE_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar o valor do nome do pacote em seu aplicativo. Para verificar se o aplicativo é válido, execute o comando `aapt debug badging` *<path to your test*

package> e tente novamente depois de encontrar o valor no nome do pacote subjacente à palavra-chave "package: name".

Durante o processo de validação de upload, o AWS Device Farm analisa o valor do nome do pacote pela saída de um comando `aapt debug badging` *<path to your package>*.

Verifique se você consegue executar esse comando com êxito em seu aplicativo Android e encontrar o valor do nome do pacote. No exemplo a seguir, o nome do pacote é app-debug.apk.

- Copie o pacote de aplicativos para seu diretório de trabalho e execute o seguinte comando:

```
$ aapt debug badging app-debug.apk | grep "package: name="
```

Um pacote de aplicativos Android válido deve gerar um resultado semelhante ao seguinte:

```
package: name='com.amazon.aws.adf.android.referenceapp' versionCode='1'
 versionName='1.0' platformBuildVersionName='5.1.1-1819727'
```

Para obter mais informações, consulte Testes de Android no AWS Device Farm.

ANDROID_APP_SDK_VERSION_VALUE_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar o valor da versão do SDK em seu aplicativo. Para verificar se o aplicativo é válido, execute o comando `aapt debug badging` *<path to your test package>* e tente novamente depois de encontrar o valor de versão do SDK subjacente à palavra-chave `sdkVersion`.

Durante o processo de validação de upload, o AWS Device Farm analisa o valor da versão do SDK pela saída de um comando `aapt debug badging` *<path to your package>*.

Verifique se você consegue executar esse comando com êxito em seu aplicativo Android e encontrar o valor do nome do pacote. No exemplo a seguir, o nome do pacote é app-debug.apk.

- Copie o pacote de aplicativos para seu diretório de trabalho e execute o seguinte comando:

```
$ aapt debug badging app-debug.apk | grep "sdkVersion"
```

Um pacote de aplicativos Android válido deve gerar um resultado semelhante ao seguinte:

```
sdkVersion:'9'
```

Para obter mais informações, consulte Testes de Android no AWS Device Farm.

ANDROID_APP_AAPT_DUMP_XMLTREE_FAILED

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar o AndroidManifest arquivo.xml válido em seu aplicativo. Para verificar se o pacote de testes é válido, execute o comando `aapt dump xmltree` *`<path to your test package>`* `AndroidManifest.xml` e tente novamente se o comando não imprimir nenhum erro.

Durante o processo de validação de upload, o AWS Device Farm analisa as informações da árvore de análise XML para um arquivo XML contido no pacote usando o comando `aapt dump xmltree` *`<path to your package>`* `AndroidManifest.xml`.

Verifique se você consegue executar esse comando com êxito em seu aplicativo Android. No exemplo a seguir, o nome do pacote é app-debug.apk.

- Copie o pacote de aplicativos para seu diretório de trabalho e execute o seguinte comando:

```
$ aapt dump xmltree app-debug.apk. AndroidManifest.xml
```

Um pacote de aplicativos Android válido deve gerar um resultado semelhante ao seguinte:

```
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")
```

Para obter mais informações, consulte Testes de Android no AWS Device Farm.

ANDROID_APP_DEVICE_ADMIN_PERMISSIONS

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> ⚠ Warning
>
> Descobrimos que seu aplicativo requer permissões de administrador do dispositivo. Verifique se as permissões não são necessárias executando o comando `aapt dump xmltree` *`<path to your test package>`* `AndroidManifest.xml` e tente novamente assim que confirmar que a saída não contém a palavra-chave `android.permission.BIND_DEVICE_ADMIN`.

Durante o processo de validação de upload, o AWS Device Farm analisa as informações de permissão da árvore de análise xml para um arquivo xml contido no pacote usando o comando `aapt dump xmltree` *`<path to your package>`* `AndroidManifest.xml`.

Descobrimos que seu aplicativo não requer permissão de administrador do dispositivo. No exemplo a seguir, o nome do pacote é app-debug.apk.

- Copie o pacote de aplicativos para seu diretório de trabalho e execute o seguinte comando:

```
$ aapt dump xmltree app-debug.apk AndroidManifest.xml
```

Você provavelmente chegará a um resultado como o seguinte:

```
N: android=http://schemas.android.com/apk/res/android
  E: manifest (line=2)
    A: android:versionCode(0x0101021b)=(type 0x10)0x1
    A: android:versionName(0x0101021c)="1.0" (Raw: "1.0")
    A: package="com.amazonaws.devicefarm.android.referenceapp" (Raw:
 "com.amazonaws.devicefarm.android.referenceapp")
    A: platformBuildVersionCode=(type 0x10)0x16 (Raw: "22")
    A: platformBuildVersionName="5.1.1-1819727" (Raw: "5.1.1-1819727")
    E: uses-sdk (line=7)
      A: android:minSdkVersion(0x0101020c)=(type 0x10)0xa
      A: android:targetSdkVersion(0x01010270)=(type 0x10)0x16
    E: uses-permission (line=11)
      A: android:name(0x01010003)="android.permission.INTERNET" (Raw:
 "android.permission.INTERNET")
    E: uses-permission (line=12)
      A: android:name(0x01010003)="android.permission.CAMERA" (Raw:
 "android.permission.CAMERA")
        ……
```

Se o aplicativo Android for válido, a saída não deverá conter o seguinte: `A: android:name(0x01010003)="android.permission.BIND_DEVICE_ADMIN" (Raw: "android.permission.BIND_DEVICE_ADMIN")`.

Para obter mais informações, consulte Testes de Android no AWS Device Farm.

Certas janelas do meu aplicativo Android mostram uma tela em branco ou preta

Se você estiver testando um aplicativo Android e perceber que determinadas janelas do aplicativo aparecem com uma tela preta na gravação de vídeo do teste do Device Farm, seu aplicativo pode estar usando o recurso `FLAG_SECURE` do Android. Esse sinalizador (conforme descrito na documentação oficial do Android) é usado para impedir que determinadas janelas de uma aplicação sejam gravadas por ferramentas de gravação de tela. Como resultado, o recurso de gravação de tela do Device Farm (para testes de automação e acesso remoto) pode mostrar uma tela preta no lugar da janela da sua aplicação se a janela usar esse sinalizador.

Esse sinalizador é frequentemente usado por desenvolvedores para páginas em suas aplicações que contêm informações sensíveis, como páginas de login. Se você estiver vendo uma tela preta no lugar da tela da sua aplicação para determinadas páginas, como a página de login, trabalhe com seus desenvolvedores para obter uma versão da aplicação que não use esse sinalizador para testes.

Além disso, observe que o Device Farm ainda pode interagir com janelas de aplicações que tenham esse sinalizador. Portanto, se a página de login da aplicação aparecer como uma tela preta, você ainda poderá inserir suas credenciais para fazer login na aplicação (e, assim, visualizar páginas não bloqueadas pelo sinalizador `FLAG_SECURE`).

Solução de problemas de JUnit testes do Appium Java no AWS Device Farm

O tópico a seguir lista as mensagens de erro que ocorrem durante o upload dos JUnit testes do Appium Java e recomenda soluções alternativas para resolver cada erro.

> **Note**
>
> As instruções a seguir baseiam-se no Linux x86_64 e Mac.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não conseguimos abrir seu arquivo de teste ZIP. Verifique se o arquivo é válido e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip zip-with-dependencies.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Um JUnit pacote Appium Java válido deve produzir uma saída como a seguinte:

```
.
|— 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
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_DEPENDENCY_DIR_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar o diretório dependency-jars em seu pacote de testes. Descompacte o pacote de testes, verifique se o diretório dependency-jars encontra-se no pacote e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip zip-with-dependencies.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o JUnit pacote Appium Java for válido, você encontrará o *dependency-jars* diretório dentro do diretório de trabalho:

```
.
|— 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
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DI

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível localizar um arquivo JAR na árvore do diretório dependency-jars. Descompacte o pacote de testes e abra o diretório dependency-jars, verifique se pelo menos um arquivo JAR encontra-se no diretório e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip zip-with-dependencies.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o JUnit pacote Appium Java for válido, você encontrará pelo menos um *jar* arquivo dentro do *dependency-jars* diretório:

```
.
|— 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
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar um arquivo * -tests.jar em seu pacote de testes. Descompacte o pacote de testes, verifique se pelo menos um arquivo * -tests.jar encontra-se no pacote e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip zip-with-dependencies.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o JUnit pacote Appium Java for válido, você encontrará pelo menos um *jar* arquivo, como *acme-android-appium-1.0-SNAPSHOT-tests.jar* no nosso exemplo. O nome do arquivo pode ser diferente, mas deve terminar com*-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
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JA

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível localizar um arquivo de classe no arquivo de testes JAR. Descompacte o pacote de testes e o arquivo de testes JAR, verifique se pelo menos um arquivo de classe encontra-se no arquivo JAR e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip zip-with-dependencies.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Você deve encontrar pelo menos um arquivo jar, como *acme-android-appium-1.0-SNAPSHOT-tests.jar* no nosso exemplo. O nome do arquivo pode ser diferente, mas deve terminar com*-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. Assim que conseguir extrair os arquivos, deverá encontrar pelo menos uma classe na árvore do diretório de trabalho executando o comando:

```
$ tree .
```

Você deve ver um resultado semelhante a este:

```
.
|— 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
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_JUNIT_VERSION_VALUE_UNKNOWN

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar um valor de JUnit versão. Descompacte seu pacote de teste e abra o diretório dependency-jars, verifique se o arquivo JUnit JAR está dentro do diretório e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   tree .
   ```

   O resultado deve ser semelhante ao seguinte:

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

```
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
    |— junit-4.10.jar
    |— com.some-dependency.bar-4.1.jar
    |— com.another-dependency.thing-1.0.jar
    |— joda-time-2.7.jar
    `— log4j-1.2.14.jar
```

Se o JUnit pacote Appium Java for válido, você encontrará o arquivo de JUnit dependência semelhante ao arquivo jar *junit-4.10.jar* em nosso exemplo. O nome deve consistir na palavra-chave *junit* e em seu número de versão, que neste exemplo é 4.10.

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Descobrimos que a JUnit versão era inferior à versão mínima 4.10 que suportamos. Altere a JUnit versão e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Você deve encontrar um arquivo de JUnit dependência como *junit-4.10.jar* em nosso exemplo e seu número de versão, que em nosso exemplo é 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**
>
> Seus testes podem não ser executados corretamente se a JUnit versão especificada em seu pacote de teste for inferior à versão mínima 4.10 que suportamos.

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

Solução de problemas de testes de aplicativos JUnit web Appium Java no AWS Device Farm

O tópico a seguir lista as mensagens de erro que ocorrem durante o upload dos testes do aplicativo Appium Java JUnit Web e recomenda soluções alternativas para resolver cada erro. Para obter mais informações sobre como usar o Appium com o Device Farm, consulte the section called “Appium”.

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não conseguimos abrir seu arquivo de teste ZIP. Verifique se o arquivo é válido e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip zip-with-dependencies.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

   Um JUnit pacote Appium Java válido deve produzir uma saída como a seguinte:

```
.
|— 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

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar o diretório dependency-jars em seu pacote de testes. Descompacte o pacote de testes, verifique se o diretório dependency-jars encontra-se no pacote e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip zip-with-dependencies.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o JUnit pacote Appium Java for válido, você encontrará o *dependency-jars* diretório dentro do diretório de trabalho:

```
.
|— 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

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível localizar um arquivo JAR na árvore do diretório dependency-jars. Descompacte o pacote de testes e abra o diretório dependency-jars, verifique se pelo menos um arquivo JAR encontra-se no diretório e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip zip-with-dependencies.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o JUnit pacote Appium Java for válido, você encontrará pelo menos um *jar* arquivo dentro do *dependency-jars* diretório:

```
.
|— 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

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar um arquivo * -tests.jar em seu pacote de testes. Descompacte o pacote de testes, verifique se pelo menos um arquivo * -tests.jar encontra-se no pacote e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip zip-with-dependencies.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o JUnit pacote Appium Java for válido, você encontrará pelo menos um *jar* arquivo, como *acme-android-appium-1.0-SNAPSHOT-tests.jar* no nosso exemplo. O nome do arquivo pode ser diferente, mas deve terminar com*-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_TES

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível localizar um arquivo de classe no arquivo de testes JAR. Descompacte o pacote de testes e o arquivo de testes JAR, verifique se pelo menos um arquivo de classe encontra-se no arquivo JAR e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip zip-with-dependencies.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Você deve encontrar pelo menos um arquivo jar, como *acme-android-appium-1.0-SNAPSHOT-tests.jar* no nosso exemplo. O nome do arquivo pode ser diferente, mas deve terminar com*-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. Assim que conseguir extrair os arquivos, deverá encontrar pelo menos uma classe na árvore do diretório de trabalho executando o comando:

```
$ tree .
```

Você deve ver um resultado semelhante a este:

```
.
|— 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

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar um valor de JUnit versão. Descompacte seu pacote de teste e abra o diretório dependency-jars, verifique se o arquivo JUnit JAR está dentro do diretório e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   tree .
   ```

   O resultado deve ser semelhante ao seguinte:

   ```
   .
   |– acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
   |– acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
   |– zip-with-dependencies.zip (this .zip file contains all of the items)
   `– dependency-jars  (this is the directory that contains all of your dependencies,
    built as JAR files)
         |– junit-4.10.jar
   ```

```
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

Se o JUnit pacote Appium Java for válido, você encontrará o arquivo de JUnit dependência semelhante ao arquivo jar *junit-4.10.jar* em nosso exemplo. O nome deve consistir na palavra-chave *junit* e em seu número de versão, que neste exemplo é 4.10.

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Descobrimos que a JUnit versão era inferior à versão mínima 4.10 que suportamos. Altere a JUnit versão e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Você deve encontrar um arquivo de JUnit dependência como *junit-4.10.jar* em nosso exemplo e seu número de versão, que em nosso exemplo é 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**
>
> Seus testes podem não ser executados corretamente se a JUnit versão especificada em seu pacote de teste for inferior à versão mínima 4.10 que suportamos.

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

Solução de problemas de testes Appium Java TestNG no AWS Device Farm

O tópico a seguir lista mensagens de erro que ocorrem durante o upload de testes do Appium Java TestNG e recomenda soluções para resolver cada erro.

> **Note**
>
> As instruções a seguir baseiam-se no Linux x86_64 e Mac.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não conseguimos abrir seu arquivo de teste ZIP. Verifique se o arquivo é válido e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip zip-with-dependencies.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Um JUnit pacote Appium Java válido deve produzir uma saída como a seguinte:

```
.
|— 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
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar o diretório `dependency-jars` em seu pacote de testes. Descompacte o pacote de testes, verifique se o diretório `dependency-jars` encontra-se no pacote e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip zip-with-dependencies.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o JUnit pacote Appium Java for válido, você encontrará o *dependency-jars* diretório dentro do diretório de trabalho.

```
.
|— 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
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível localizar um arquivo JAR na árvore do diretório dependency-jars. Descompacte o pacote de testes e abra o diretório dependency-jars, verifique se pelo menos um arquivo JAR encontra-se no diretório e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip zip-with-dependencies.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o JUnit pacote Appium Java for válido, você encontrará pelo menos um *jar* arquivo dentro do *dependency-jars* diretório.

```
.
|— 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
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar um arquivo * -tests.jar em seu pacote de testes. Descompacte o pacote de testes, verifique se pelo menos um arquivo * -tests.jar encontra-se no pacote e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip zip-with-dependencies.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o JUnit pacote Appium Java for válido, você encontrará pelo menos um *jar* arquivo, como *acme-android-appium-1.0-SNAPSHOT-tests.jar* no nosso exemplo. O nome do arquivo pode ser diferente, mas deve terminar com*-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
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível localizar um arquivo de classe no arquivo de testes JAR. Descompacte o pacote de testes e o arquivo de testes JAR, verifique se pelo menos um arquivo de classe encontra-se no arquivo JAR e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip zip-with-dependencies.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

   Você deve encontrar pelo menos um arquivo jar, como *acme-android-appium-1.0-SNAPSHOT-tests.jar* no nosso exemplo. O nome do arquivo pode ser diferente, mas deve terminar com*-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. Para extrair arquivos do arquivo jar, você pode executar o seguinte comando:

```
$ jar xf acme-android-appium-1.0-SNAPSHOT-tests.jar
```

4. Assim que conseguir extrair os arquivos, execute o seguinte comando:

```
$ tree .
```

   Você deve encontrar pelo menos uma classe na árvore do diretório de trabalho:

```
.
|— 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
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

Solução de problemas de aplicações Web Appium Java TestNG no AWS Device Farm

O tópico a seguir lista mensagens de erro que ocorrem durante o upload de testes de aplicativos web do Appium Java TestNG e recomenda soluções para resolver cada erro.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não conseguimos abrir seu arquivo de teste ZIP. Verifique se o arquivo é válido e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Um JUnit pacote Appium Java válido deve produzir uma saída como a seguinte:

```
.
|— 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
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSIN

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar o diretório dependency-jars em seu pacote de testes. Descompacte o pacote de testes, verifique se o diretório dependency-jars encontra-se no pacote e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o JUnit pacote Appium Java for válido, você encontrará o *dependency-jars* diretório dentro do diretório de trabalho.

```
.
|— 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
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPEND

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível localizar um arquivo JAR na árvore do diretório dependency-jars. Descompacte o pacote de testes e abra o diretório dependency-jars, verifique se pelo menos um arquivo JAR encontra-se no diretório e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o JUnit pacote Appium Java for válido, você encontrará pelo menos um *jar* arquivo dentro do *dependency-jars* diretório.

```
.
|— 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
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar um arquivo * -tests.jar em seu pacote de testes. Descompacte o pacote de testes, verifique se pelo menos um arquivo * -tests.jar encontra-se no pacote e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o JUnit pacote Appium Java for válido, você encontrará pelo menos um *jar* arquivo, como *acme-android-appium-1.0-SNAPSHOT-tests.jar* no nosso exemplo. O nome do arquivo pode ser diferente, mas deve terminar com*-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
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_T

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível localizar um arquivo de classe no arquivo de testes JAR. Descompacte o pacote de testes e o arquivo de testes JAR, verifique se pelo menos um arquivo de classe encontra-se no arquivo JAR e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip zip-with-dependencies.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Você deve encontrar pelo menos um arquivo jar, como *acme-android-appium-1.0-SNAPSHOT-tests.jar* no nosso exemplo. O nome do arquivo pode ser diferente, mas deve terminar com*-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. Para extrair arquivos do arquivo jar, você pode executar o seguinte comando:

```
$ jar xf acme-android-appium-1.0-SNAPSHOT-tests.jar
```

4. Assim que conseguir extrair os arquivos, execute o seguinte comando:

```
$ tree .
```

Você deve encontrar pelo menos uma classe na árvore do diretório de trabalho:

```
.
|— 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
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

Solução de problemas de testes Appium Python no AWS Device Farm

O tópico a seguir lista mensagens de erro que ocorrem durante o upload de testes do Appium Python e recomenda soluções para resolver cada erro.

APPIUM_PYTHON_TEST_PACKAGE_UNZIP_FAILED

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não conseguimos abrir o arquivo de teste ZIP do Appium. Verifique se o arquivo é válido e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip test_bundle.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Um pacote válido do Appium Python deve gerar um resultado semelhante ao seguinte:

```
.
|-- 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
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível localizar um arquivo de dependência wheel na árvore do diretório wheelhouse. Descompacte o pacote de testes e abra o diretório wheelhouse, verifique se pelo menos um arquivo wheel encontra-se no diretório e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip test_bundle.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

Se o pacote Appium Python for válido, você encontrará pelo menos um arquivo *.whl* dependente, como os arquivos destacados, dentro do diretório. *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
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_INVALID_PLATFORM

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Encontramos pelo menos um arquivo wheel que especificou uma plataforma para a qual não oferecemos compatibilidade. Descompacte seu pacote de testes e abra o diretório wheelhouse, verifique se os nomes dos arquivos wheel terminam com -any.whl ou -linux_x86_64.whl e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip test_bundle.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o pacote Appium Python for válido, você encontrará pelo menos um arquivo *.whl* dependente, como os arquivos destacados, dentro do diretório. *wheelhouse* O nome do arquivo pode ser diferente, mas deve terminar com *-any.whl* ou*-linux_x86_64.whl*, que especifica a plataforma. Não há suporte para outras plataformas como `windows`.

```
.
|-- 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
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar o diretório tests em seu pacote de testes. Descompacte o pacote de testes, verifique se o diretório tests está no pacote e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip test_bundle.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o pacote Appium Python for válido, você encontrará o *tests* diretório dentro do diretório de trabalho.

```
.
|-- 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
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível localizar um arquivo de teste válido na árvore do diretório tests. Descompacte seu pacote de testes e abra o diretório tests, verifique se pelo menos um nome de arquivo começa ou termina com a palavra-chave "test" e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip test_bundle.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o pacote Appium Python for válido, você encontrará o *tests* diretório dentro do diretório de trabalho. O nome do arquivo pode ser diferente, mas ele deve começar com *test_* ou terminar com*_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
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar o arquivo requirements.txt em seu pacote de testes. Descompacte o pacote de testes, verifique se o arquivo requirements.txt encontra-se no pacote e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip test_bundle.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o pacote Appium Python for válido, você encontrará o *requirements.txt* arquivo dentro do diretório de trabalho.

```
.
|-- 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
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Percebemos que a versão pytest era anterior à versão mínima compatível 2.8.0. Altere a versão pytest no arquivo requirements.txt e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip test_bundle.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Você deve encontrar o *requirements.txt* arquivo dentro do diretório de trabalho.

```
.
|-- 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. Para obter a versão pytest, você pode executar o seguinte comando:

```
$ grep "pytest" requirements.txt
```

Você provavelmente chegará a um resultado como o seguinte:

```
pytest==2.9.0
```

Ele mostra a versão do pytest que, neste exemplo, é 2.9.0. Se o pacote do Appium Python for válido, a versão de pytest deverá ser posterior ou igual a 2.8.0.

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS_FAIL

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível instalar as wheels de dependência. Descompacte seu pacote de testes e abra o arquivo requirements.txt e o diretório wheelhouse, verifique se as wheels de dependência especificadas no arquivo requirements.txt correspondem exatamente às wheels de dependência no diretório wheelhouse e tente novamente.

É altamente recomendável configurar o virtualenv do Python para testes de empacotamento. Veja aqui um exemplo de fluxo de criação de um ambiente virtual por meio do virtualenv do Python e, em seguida, de sua ativação:

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

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip test_bundle.zip
   ```

2. Para testar os arquivos wheel de instalação, execute o seguinte comando:

   ```
   $ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
   requirements.txt
   ```

   Um pacote válido do Appium Python deve gerar um resultado semelhante ao seguinte:

   ```
   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. Para desativar o ambiente virtual, você pode executar o seguinte comando:

```
$ deactivate
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível coletar testes no diretório tests. Descompacte o pacote de testes. Para verificar se o pacote de testes é válido, execute o comando `py.test --collect-only <path to your tests directory>` e tente novamente se o comando não imprimir nenhum erro.

É altamente recomendável configurar o virtualenv do Python para testes de empacotamento. Veja aqui um exemplo de fluxo de criação de um ambiente virtual por meio do virtualenv do Python e, em seguida, de sua ativação:

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

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip test_bundle.zip
```

2. Para instalar os arquivos wheel, execute o seguinte comando:

```
$ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
requirements.txt
```

3. Para coletar os testes, execute o seguinte comando:

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

Um pacote válido do Appium Python deve gerar um resultado semelhante ao seguinte:

```
==================== test session starts ====================
platform darwin -- Python 2.7.11, pytest-2.9.0, py-1.4.31, pluggy-0.3.1
rootdir: /Users/zhena/Desktop/Ios/tests, inifile:
collected 1 items
<Module 'test_unittest.py'>
  <UnitTestCase 'DeviceFarmAppiumWebTests'>
    <TestCaseFunction 'test_devicefarm'>

==================== no tests ran in 0.11 seconds ====================
```

4. Para desativar o ambiente virtual, você pode executar o seguinte comando:

```
$ deactivate
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEELS_INSUFFICIENT

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar dependências de wheel suficientes no diretório wheelhouse. Descompacte seu pacote de teste e, em seguida, abra o diretório wheelhouse. Verifique se você tem todas as dependências de wheel especificadas no arquivo requirements.txt.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip test_bundle.zip
```

2. Verifique o tamanho do *requirements.txt* arquivo, bem como o número de arquivos *.whl* dependentes no diretório do wheelhouse:

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

   Se o número de arquivos *.whl* dependentes for menor que o número de linhas não vazias em seu *requirements.txt* arquivo, você precisará garantir o seguinte:

   - Há um arquivo *.whl* dependente correspondente a cada linha do *requirements.txt* arquivo.
   - Não há outras linhas no *requirements.txt* arquivo que contenham informações além dos nomes dos pacotes de dependências.
   - Nenhum nome de dependência é duplicado em várias linhas no *requirements.txt* arquivo, de forma que duas linhas no arquivo possam corresponder a um arquivo *.whl* dependente.

   O AWS Device Farm não suporta linhas no *requirements.txt* arquivo que não correspondam diretamente aos pacotes de dependência, como linhas que especificam opções globais para o `pip install` comando. Consulte Requirements File Format para obter uma lista de opções globais.

   Para obter mais informações, consulte Testes Appium e AWS Device Farm.

Solução de problemas de testes de aplicações Web Appium Python no AWS Device Farm

O tópico a seguir lista mensagens de erro que ocorrem durante o upload de testes de aplicativos web do Appium Python e recomenda soluções para resolver cada erro.

APPIUM_WEB_PYTHON_TEST_PACKAGE_UNZIP_FAILED

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não conseguimos abrir o arquivo de teste ZIP do Appium. Verifique se o arquivo é válido e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip test_bundle.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Um pacote válido do Appium Python deve gerar um resultado semelhante ao seguinte:

   ```
   .
   |-- 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
   ```

   Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível localizar um arquivo de dependência wheel na árvore do diretório wheelhouse. Descompacte o pacote de testes e abra o diretório wheelhouse, verifique se pelo menos um arquivo wheel encontra-se no diretório e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip test_bundle.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Se o pacote Appium Python for válido, você encontrará pelo menos um arquivo *.whl* dependente, como os arquivos destacados, dentro do diretório. *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
   ```

   Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PLATFORM

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Encontramos pelo menos um arquivo wheel que especificou uma plataforma para a qual não oferecemos compatibilidade. Descompacte seu pacote de testes e abra o diretório wheelhouse, verifique se os nomes dos arquivos wheel terminam com -any.whl ou -linux_x86_64.whl e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip test_bundle.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Se o pacote Appium Python for válido, você encontrará pelo menos um arquivo *.whl* dependente, como os arquivos destacados, dentro do diretório. *wheelhouse* O nome do arquivo pode ser diferente, mas deve terminar com *-any.whl* ou*-linux_x86_64.whl*, que especifica a plataforma. Não há suporte para outras plataformas como windows.

   ```
   .
   |-- 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
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar o diretório tests em seu pacote de testes. Descompacte o pacote de testes, verifique se o diretório tests está no pacote e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip test_bundle.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Se o pacote Appium Python for válido, você encontrará o *tests* diretório dentro do diretório de trabalho.

   ```
   .
   |-- 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
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível localizar um arquivo de teste válido na árvore do diretório tests. Descompacte seu pacote de testes e abra o diretório tests, verifique se pelo menos um nome de arquivo começa ou termina com a palavra-chave "test" e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip test_bundle.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Se o pacote Appium Python for válido, você encontrará o *tests* diretório dentro do diretório de trabalho. O nome do arquivo pode ser diferente, mas deve começar com *test_* ou terminar com*_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
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISS

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar o arquivo requirements.txt em seu pacote de testes. Descompacte o pacote de testes, verifique se o arquivo requirements.txt encontra-se no pacote e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip test_bundle.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Se o pacote Appium Python for válido, você encontrará o *`requirements.txt`* arquivo dentro do diretório de trabalho.

   ```
   .
   |-- 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
```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Percebemos que a versão pytest era anterior à versão mínima compatível 2.8.0. Altere a versão pytest no arquivo requirements.txt e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip test_bundle.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Você deve encontrar o *requirement.txt* arquivo dentro do diretório de trabalho.

   ```
   .
   |-- 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. Para obter a versão pytest, você pode executar o seguinte comando:

```
$ grep "pytest" requirements.txt
```

Você provavelmente chegará a um resultado como o seguinte:

```
pytest==2.9.0
```

Ele mostra a versão do pytest que, neste exemplo, é 2.9.0. Se o pacote do Appium Python for válido, a versão de pytest deverá ser posterior ou igual a 2.8.0.

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível instalar as wheels de dependência. Descompacte seu pacote de testes e abra o arquivo requirements.txt e o diretório wheelhouse, verifique se as wheels de dependência especificadas no arquivo requirements.txt correspondem exatamente às wheels de dependência no diretório wheelhouse e tente novamente.

É altamente recomendável configurar o virtualenv do Python para testes de empacotamento. Veja aqui um exemplo de fluxo de criação de um ambiente virtual por meio do virtualenv do Python e, em seguida, de sua ativação:

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

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip test_bundle.zip
```

2. Para testar os arquivos wheel de instalação, execute o seguinte comando:

```
$ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
requirements.txt
```

   Um pacote válido do Appium Python deve gerar um resultado semelhante ao seguinte:

```
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. Para desativar o ambiente virtual, você pode executar o seguinte comando:

```
$ deactivate
```

   Para obter mais informações, consulte Testes Appium e AWS Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível coletar testes no diretório tests. Descompacte o pacote de testes. Para verificar se o pacote de testes é válido, execute o comando <py.test --collect-only> <caminho para seu pacote de testes> e tente novamente se o comando não imprimir nenhum erro.

É altamente recomendável configurar o virtualenv do Python para testes de empacotamento. Veja aqui um exemplo de fluxo de criação de um ambiente virtual por meio do virtualenv do Python e, em seguida, de sua ativação:

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

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip test_bundle.zip
   ```

2. Para instalar os arquivos wheel, execute o seguinte comando:

   ```
   $ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
   requirements.txt
   ```

3. Para coletar os testes, execute o seguinte comando:

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

   Um pacote válido do Appium Python deve gerar um resultado semelhante ao seguinte:

   ```
   ==================== test session starts ====================
   platform darwin -- Python 2.7.11, pytest-2.9.0, py-1.4.31, pluggy-0.3.1
   rootdir: /Users/zhena/Desktop/Ios/tests, inifile:
   collected 1 items
   <Module 'test_unittest.py'>
     <UnitTestCase 'DeviceFarmAppiumWebTests'>
       <TestCaseFunction 'test_devicefarm'>

   ==================== no tests ran in 0.11 seconds ====================
   ```

4. Para desativar o ambiente virtual, você pode executar o seguinte comando:

   ```
   $ deactivate
   ```

Para obter mais informações, consulte Testes Appium e AWS Device Farm.

Solução de problemas de testes de instrumentação no AWS Device Farm

O tópico a seguir lista mensagens de erro que ocorrem durante o upload de testes de instrumentação e recomenda soluções para resolver cada erro.

> **Note**
>
> Para considerações importantes ao usar testes de instrumentação no AWS Device Farm, consulte. Instrumentação para Android e AWS Device Farm

INSTRUMENTATION_TEST_PACKAGE_UNZIP_FAILED

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

```
Warning: We could not open your test APK file. Please verify that the file is valid and
    try again.
```

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é app-debug-androidTest-unaligned.apk.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip app-debug-androidTest-unaligned.apk
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Um pacote de testes de instrumentação válido deve gerar um resultado semelhante ao seguinte:

   ```
   .
   ```

```
|-- AndroidManifest.xml
|--  classes.dex
|-- resources.arsc
|-- LICENSE-junit.txt
|-- junit (directory)
`-- META-INF (directory)
```

Para obter mais informações, consulte Instrumentação para Android e AWS Device Farm.

INSTRUMENTATION_TEST_PACKAGE_AAPT_DEBUG_BADGING_FAILED

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

```
We could not extract information about your test package. Please verify that the
      test package is valid by running the command "aapt debug badging <path to your
 test
      package>", and try again after the command does not print any error.
```

Durante o processo de validação do upload, o Device Farm analisa as informações da saída do comando `aapt debug badging <path to your package>`.

Verifique se você consegue executar esse comando com êxito em seu pacote de testes de instrumentação.

No exemplo a seguir, o nome do pacote é app-debug-androidTest-unaligned.apk.

- Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

  ```
  $ aapt debug badging app-debug-androidTest-unaligned.apk
  ```

  Um pacote de testes de instrumentação válido deve gerar um resultado semelhante ao seguinte:

  ```
  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-implied-feature: name='android.hardware.touchscreen' reason='default feature
 for all apps'
supports-screens: 'small' 'normal' 'large' 'xlarge'
supports-any-density: 'true'
locales: '--_--'
densities: '160'
```

Para obter mais informações, consulte Instrumentação para Android e AWS Device Farm.

INSTRUMENTATION_TEST_PACKAGE_INSTRUMENTATION_RUNNER_VALU

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

```
We could not find the instrumentation runner value in the AndroidManifest.xml.
      Please verify the test package is valid by running the command "aapt dump xmltree
 <path to
      your test package> AndroidManifest.xml", and try again after finding the
 instrumentation
      runner value behind the keyword "instrumentation."
```

Durante o processo de validação de upload, o Device Farm analisa o valor do executor de instrumentação da árvore de análise XML para um arquivo XML contido no pacote. Você pode usar os seguintes comandos: `aapt dump xmltree <path to your package> AndroidManifest.xml`.

Verifique se você consegue executar esse comando com êxito em seu pacote de testes de instrumentação e encontrar o valor de instrumentação.

No exemplo a seguir, o nome do pacote é app-debug-androidTest-unaligned.apk.

- Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

  ```
  $ aapt dump xmltree app-debug-androidTest-unaligned.apk AndroidManifest.xml | grep
   -A5 "instrumentation"
  ```

  Um pacote de testes de instrumentação válido deve gerar um resultado semelhante ao seguinte:

  ```
  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
```

Para obter mais informações, consulte Instrumentação para Android e AWS Device Farm.

INSTRUMENTATION_TEST_PACKAGE_AAPT_DUMP_XMLTREE_FAILED

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

```
We could not find the valid AndroidManifest.xml in your test package. Please
      verify that the test package is valid by running the command "aapt dump xmltree
 <path to
      your test package> AndroidManifest.xml", and try again after the command does not
 print any
      error.
```

Durante o processo de validação de upload, o Device Farm analisa as informações da árvore de análise XML de um arquivo XML contido no pacote usando o seguinte comando: `aapt dump xmltree <path to your package> AndroidManifest.xml`.

Verifique se você consegue executar esse comando com êxito em seu pacote de testes de instrumentação.

No exemplo a seguir, o nome do pacote é app-debug-androidTest-unaligned.apk.

- Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

  ```
  $ aapt dump xmltree app-debug-androidTest-unaligned.apk AndroidManifest.xml
  ```

  Um pacote de testes de instrumentação válido deve gerar um resultado semelhante ao seguinte:

  ```
  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")
```

Para obter mais informações, consulte Instrumentação para Android e AWS Device Farm.

INSTRUMENTATION_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISS

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

```
We could not find the package name in your test package. Please verify that the
      test package is valid by running the command "aapt debug badging <path to your
 test
      package>", and try again after finding the package name value behind the keyword
 "package:
      name."
```

Durante o processo de validação do upload, o Device Farm analisa o valor do nome do pacote pela saída do seguinte comando: `aapt debug badging <path to your package>`.

Verifique se você consegue executar esse comando com êxito em seu pacote de testes de instrumentação e encontrar o valor do nome do pacote.

No exemplo a seguir, o nome do pacote é app-debug-androidTest-unaligned.apk.

- Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

  ```
  $ aapt debug badging app-debug-androidTest-unaligned.apk | grep "package: name="
  ```

  Um pacote de testes de instrumentação válido deve gerar um resultado semelhante ao seguinte:

  ```
  package: name='com.amazon.aws.adf.android.referenceapp.test' versionCode=''
   versionName='' platformBuildVersionName='5.1.1-1819727'
  ```

  Para obter mais informações, consulte Instrumentação para Android e AWS Device Farm.

Solução de problemas de testes de aplicações iOS no AWS Device Farm

O tópico a seguir lista mensagens de erro que ocorrem durante o upload de testes de aplicativos iOS e recomenda soluções para resolver cada erro.

> **Note**
>
> As instruções a seguir baseiam-se no Linux x86_64 e Mac.

IOS_APP_UNZIP_FAILED

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não conseguimos abrir seu aplicativo. Verifique se o arquivo é válido e tente novamente.

Verifique se você consegue descompactar o pacote de aplicativos sem erros. No exemplo a seguir, o nome do pacote é AWSDeviceFarmi OSReference App.ipa.

1. Copie o pacote de aplicativos para seu diretório de trabalho e execute o seguinte comando:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Um pacote de aplicativos iOS válido deve gerar um resultado semelhante ao seguinte:

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

Para obter mais informações, consulte Testes de iOS no AWS Device Farm.

IOS_APP_PAYLOAD_DIR_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar o diretório Payload em seu pacote de aplicativos. Descompacte o pacote de aplicativos, verifique se o diretório Payload encontra-se no pacote e tente novamente.

No exemplo a seguir, o nome do pacote é AWSDeviceFarmi OSReference App.ipa.

1. Copie o pacote de aplicativos para seu diretório de trabalho e execute o seguinte comando:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o pacote do aplicativo iOS for válido, você encontrará o *Payload* diretório dentro do diretório de trabalho.

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

Para obter mais informações, consulte Testes de iOS no AWS Device Farm.

IOS_APP_APP_DIR_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar o diretório .app no diretório Payload. Descompacte o pacote de aplicativos e abra o diretório Payload, verifique se o diretório .app encontra-se no diretório e tente novamente.

No exemplo a seguir, o nome do pacote é AWSDeviceFarmi OSReference App.ipa.

1. Copie o pacote de aplicativos para seu diretório de trabalho e execute o seguinte comando:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Se o pacote do aplicativo iOS for válido, você encontrará um *.app* diretório como *AWSDeviceFarmiOSReferenceApp.app* no nosso exemplo dentro do *Payload* diretório.

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

Para obter mais informações, consulte Testes de iOS no AWS Device Farm.

IOS_APP_PLIST_FILE_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar o arquivo Info.plist no diretório .app. Descompacte o pacote de aplicativos e abra o diretório .app, verifique se o arquivo Info.plist encontra-se no diretório e tente novamente.

No exemplo a seguir, o nome do pacote é AWSDeviceFarmi OSReference App.ipa.

1. Copie o pacote de aplicativos para seu diretório de trabalho e execute o seguinte comando:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Se o pacote do aplicativo iOS for válido, você encontrará o *Info.plist* arquivo dentro do *.app* diretório, como *AWSDeviceFarmiOSReferenceApp.app* no nosso exemplo.

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

Para obter mais informações, consulte Testes de iOS no AWS Device Farm.

IOS_APP_CPU_ARCHITECTURE_VALUE_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar o valor de arquitetura da CPU no arquivo Info.plist. Descompacte seu aplicativo e abra o arquivo Info.plist dentro do diretório.app, verifique se a chave "UIRequiredDeviceCapabilities" está especificada e tente novamente.

No exemplo a seguir, o nome do pacote é AWSDeviceFarmi OSReference App.ipa.

1. Copie o pacote de aplicativos para seu diretório de trabalho e execute o seguinte comando:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

   Você deve encontrar o *Info.plist* arquivo dentro de um *.app* diretório, como *AWSDeviceFarmiOSReferenceApp.app* no nosso exemplo:

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

3. Para encontrar o valor de arquitetura da CPU, você pode abrir o Info.plist usando o Xcode ou Python.

   Para o Python, você pode instalar o módulo biplist executando o seguinte comando:

```
$ pip install biplist
```

4. Em seguida, abra o Python e execute o seguinte comando:

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['UIRequiredDeviceCapabilities']
```

Um pacote de aplicativos iOS válido deve gerar um resultado semelhante ao seguinte:

```
['armv7']
```

Para obter mais informações, consulte Testes de iOS no AWS Device Farm.

IOS_APP_PLATFORM_VALUE_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar o valor da plataforma no arquivo Info.plist. Descompacte seu aplicativo e abra o arquivo Info.plist dentro do diretório.app, verifique se a chave "CFBundleSupportedPlatforms" está especificada e tente novamente.

No exemplo a seguir, o nome do pacote é AWSDeviceFarmi OSReference App.ipa.

1. Copie o pacote de aplicativos para seu diretório de trabalho e execute o seguinte comando:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Você deve encontrar o *Info.plist* arquivo dentro de um *.app* diretório, como *AWSDeviceFarmiOSReferenceApp.app* no nosso exemplo:

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

3. Para encontrar o valor da plataforma, você pode abrir o Info.plist usando o Xcode ou Python.

   Para o Python, você pode instalar o módulo biplist executando o seguinte comando:

   ```
   $ pip install biplist
   ```

4. Em seguida, abra o Python e execute o seguinte comando:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
   Info.plist')
   print info_plist['CFBundleSupportedPlatforms']
   ```

   Um pacote de aplicativos iOS válido deve gerar um resultado semelhante ao seguinte:

   ```
   ['iPhoneOS']
   ```

   Para obter mais informações, consulte Testes de iOS no AWS Device Farm.

IOS_APP_WRONG_PLATFORM_DEVICE_VALUE

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Descobrimos que o valor do dispositivo de plataforma estava errado no arquivo Info.plist. Descompacte seu aplicativo e abra o arquivo Info.plist dentro do diretório.app, verifique se o valor da chave "CFBundleSupportedPlatforms" não contém a palavra-chave “simulador” e tente novamente.

No exemplo a seguir, o nome do pacote é AWSDeviceFarmi OSReference App.ipa.

1. Copie o pacote de aplicativos para seu diretório de trabalho e execute o seguinte comando:

```
$ unzip AWSDeviceFarmiOSReferenceApp.ipa
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Você deve encontrar o *Info.plist* arquivo dentro de um *.app* diretório, como *AWSDeviceFarmiOSReferenceApp.app* no nosso exemplo:

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

3. Para encontrar o valor da plataforma, você pode abrir o Info.plist usando o Xcode ou Python.

Para o Python, você pode instalar o módulo biplist executando o seguinte comando:

```
$ pip install biplist
```

4. Em seguida, abra o Python e execute o seguinte comando:

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['CFBundleSupportedPlatforms']
```

Um pacote de aplicativos iOS válido deve gerar um resultado semelhante ao seguinte:

```
['iPhoneOS']
```

Se o pacote de aplicativos iOS for válido, o valor não deve conter a palavra-chave `simulator`.

Para obter mais informações, consulte Testes de iOS no AWS Device Farm.

IOS_APP_FORM_FACTOR_VALUE_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar o valor do formato no arquivo Info.plist. Descompacte seu aplicativo e abra o arquivo Info.plist dentro do diretório.app, verifique se a chave "UIDeviceFamília" está especificada e tente novamente.

No exemplo a seguir, o nome do pacote é AWSDeviceFarmi OSReference App.ipa.

1. Copie o pacote de aplicativos para seu diretório de trabalho e execute o seguinte comando:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Você deve encontrar o *Info.plist* arquivo dentro de um *.app* diretório, como *AWSDeviceFarmiOSReferenceApp.app* no nosso exemplo:

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

3. Para encontrar o valor do formato, você pode abrir o Info.plist usando o Xcode ou Python.

   Para o Python, você pode instalar o módulo biplist executando o seguinte comando:

   ```
   $ pip install biplist
   ```

4. Em seguida, abra o Python e execute o seguinte comando:

   ```
   import biplist
   ```

```
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['UIDeviceFamily']
```

Um pacote de aplicativos iOS válido deve gerar um resultado semelhante ao seguinte:

```
[1, 2]
```

Para obter mais informações, consulte Testes de iOS no AWS Device Farm.

IOS_APP_PACKAGE_NAME_VALUE_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar o valor do nome do pacote no arquivo Info.plist. Descompacte seu aplicativo e abra o arquivo Info.plist dentro do diretório.app, verifique se a chave "CFBundleIdentifier" está especificada e tente novamente.

No exemplo a seguir, o nome do pacote é AWSDeviceFarmi OSReference App.ipa.

1. Copie o pacote de aplicativos para seu diretório de trabalho e execute o seguinte comando:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Você deve encontrar o *Info.plist* arquivo dentro de um *.app* diretório, como *AWSDeviceFarmiOSReferenceApp.app* no nosso exemplo:

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

```
                        `-- (any other files)
```

3. Para encontrar o valor do nome do pacote, você pode abrir o Info.plist usando o Xcode ou Python.

   Para o Python, você pode instalar o módulo biplist executando o seguinte comando:

   ```
   $ pip install biplist
   ```

4. Em seguida, abra o Python e execute o seguinte comando:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
   Info.plist')
   print info_plist['CFBundleIdentifier']
   ```

   Um pacote de aplicativos iOS válido deve gerar um resultado semelhante ao seguinte:

   ```
   Amazon.AWSDeviceFarmiOSReferenceApp
   ```

   Para obter mais informações, consulte Testes de iOS no AWS Device Farm.

IOS_APP_EXECUTABLE_VALUE_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar o valor do executável no arquivo Info.plist. Descompacte seu aplicativo e abra o arquivo Info.plist dentro do diretório.app, verifique se a chave "CFBundleExecutável" está especificada e tente novamente.

No exemplo a seguir, o nome do pacote é AWSDeviceFarmi OSReference App.ipa.

1. Copie o pacote de aplicativos para seu diretório de trabalho e execute o seguinte comando:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Você deve encontrar o *Info.plist* arquivo dentro de um *.app* diretório, como *AWSDeviceFarmiOSReferenceApp.app* no nosso exemplo:

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

3. Para encontrar o valor do executável, você pode abrir o Info.plist usando o Xcode ou Python.

Para o Python, você pode instalar o módulo biplist executando o seguinte comando:

```
$ pip install biplist
```

4. Em seguida, abra o Python e execute o seguinte comando:

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['CFBundleExecutable']
```

Um pacote de aplicativos iOS válido deve gerar um resultado semelhante ao seguinte:

```
AWSDeviceFarmiOSReferenceApp
```

Para obter mais informações, consulte Testes de iOS no AWS Device Farm.

XCTest Testes de solução de problemas no AWS Device Farm

O tópico a seguir lista as mensagens de erro que ocorrem durante o upload dos XCTest testes e recomenda soluções alternativas para resolver cada erro.

> **Note**
>
> As instruções a seguir presumem que você está usando o MacOS.

XCTEST_TEST_PACKAGE_UNZIP_FAILED

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não conseguimos abrir seu arquivo de teste ZIP. Verifique se o arquivo é válido e tente novamente.

Verifique se você consegue descompactar o pacote de aplicativos sem erros. No exemplo a seguir, o nome do pacote é swiftExampleTests.xctest-1.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swiftExampleTests.xctest-1.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Um XCTest pacote válido deve produzir uma saída como a seguinte:

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

   Para obter mais informações, consulte Integrando o Device Farm com XCTest o iOS.

XCTEST_TEST_PACKAGE_XCTEST_DIR_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar o diretório .xctest em seu pacote de testes. Descompacte o pacote de testes, verifique se o diretório .xctest encontra-se no pacote e tente novamente.

No exemplo a seguir, o nome do pacote é swiftExampleTests.xctest-1.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swiftExampleTests.xctest-1.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Se o XCTest pacote for válido, você encontrará um diretório com um nome semelhante ao *`swiftExampleTests.xctest`* do diretório de trabalho. O nome deve terminar com *`.xctest`*.

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

   Para obter mais informações, consulte Integrando o Device Farm com XCTest o iOS.

XCTEST_TEST_PACKAGE_PLIST_FILE_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar o arquivo Info.plist no diretório .xctest. Descompacte o pacote de testes e abra o diretório .xctest, verifique se o arquivo Info.plist encontra-se no diretório e tente novamente.

No exemplo a seguir, o nome do pacote é swiftExampleTests.xctest-1.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o XCTest pacote for válido, você encontrará o *Info.plist* arquivo dentro do *.xctest* diretório. Em nosso exemplo abaixo, o diretório é chamado*swiftExampleTests.xctest*.

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

Para obter mais informações, consulte Integrando o Device Farm com XCTest o iOS.

XCTEST_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar o valor do nome do pacote no arquivo Info.plist. Descompacte seu pacote de teste e abra o arquivo Info.plist, verifique se a chave "CFBundleIdentifier” está especificada e tente novamente.

No exemplo a seguir, o nome do pacote é swiftExampleTests.xctest-1.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Você deve encontrar o *Info.plist* arquivo dentro de um *.xctest* diretório, como *swiftExampleTests.xctest* no nosso exemplo:

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

3. Para encontrar o valor do nome do pacote, você pode abrir o Info.plist usando o Xcode ou Python.

Para o Python, você pode instalar o módulo biplist executando o seguinte comando:

```
$ pip install biplist
```

4. Em seguida, abra o Python e execute o seguinte comando:

```
import biplist
info_plist = biplist.readPlist('swiftExampleTests.xctest/Info.plist')
print info_plist['CFBundleIdentifier']
```

Um pacote de XCtest aplicativo válido deve produzir uma saída como a seguinte:

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

Para obter mais informações, consulte Integrando o Device Farm com XCTest o iOS.

XCTEST_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar o valor do executável no arquivo Info.plist. Descompacte o pacote de teste e abra o arquivo Info.plist, verifique se a chave "CFBundleExecutável" está especificada e tente novamente.

No exemplo a seguir, o nome do pacote é swiftExampleTests.xctest-1.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

   Você deve encontrar o *Info.plist* arquivo dentro de um *.xctest* diretório, como *swiftExampleTests.xctest* no nosso exemplo:

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

3. Para encontrar o valor do nome do pacote, você pode abrir o Info.plist usando o Xcode ou Python.

   Para o Python, você pode instalar o módulo biplist executando o seguinte comando:

```
$ pip install biplist
```

4. Em seguida, abra o Python e execute o seguinte comando:

```
import biplist
info_plist = biplist.readPlist('swiftExampleTests.xctest/Info.plist')
print info_plist['CFBundleExecutable']
```

   Um pacote de XCtest aplicativo válido deve produzir uma saída como a seguinte:

```
swiftExampleTests
```

Para obter mais informações, consulte Integrando o Device Farm com XCTest o iOS.

Solução de problemas de testes de XCTest interface do usuário no AWS Device Farm

O tópico a seguir lista as mensagens de erro que ocorrem durante o upload dos testes de XCTest interface do usuário e recomenda soluções alternativas para resolver cada erro.

> **Note**
>
> As instruções a seguir baseiam-se no Linux x86_64 e Mac.

XCTEST_UI_TEST_PACKAGE_UNZIP_FAILED

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

`We could not open your test IPA file. Please verify that the file is valid and try again.`

Verifique se você consegue descompactar o pacote de aplicativos sem erros. No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Um pacote de aplicativos iOS válido deve gerar um resultado semelhante ao seguinte:

   ```
   .
   ```

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

Para obter mais informações, consulte Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

```
We could not find the Payload directory inside your test package. Please unzip your test package, verify that the Payload directory is inside the package, and try again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Se o pacote de XCTest interface do usuário for válido, você encontrará o *Payload* diretório dentro do diretório de trabalho.

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

```
                    |                       |-- Info.plist
                    |                       `-- (any other files)
                    `-- (any other files)
```

Para obter mais informações, consulte Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_APP_DIR_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

```
We could not find the .app directory inside the Payload directory. Please unzip your test package and then open the Payload directory, verify that the .app directory is inside the directory, and try again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip swift-sample-UI.ipa
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o pacote de XCTest interface do usuário for válido, você encontrará um *.app* diretório como *swift-sampleUITests-Runner.app* no nosso exemplo dentro do *Payload* diretório.

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

Para obter mais informações, consulte Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_PLUGINS_DIR_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

```
We could not find the Plugins directory inside the .app directory. Please unzip your test package and then open the .app directory, verify that the Plugins directory is inside the directory, and try again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Se o pacote de XCTest interface do usuário for válido, você encontrará o *Plugins* diretório dentro de um *.app* diretório. Em nosso exemplo, o diretório é chamado*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)
   ```

   Para obter mais informações, consulte Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_XCTEST_DIR_MISSING_IN_PLUGINS_DIR

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

```
We could not find the .xctest directory inside the plugins directory.
Please unzip your test package and then open the plugins directory, verify
that the .xctest directory is inside the directory, and try again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Se o pacote de XCTest interface do usuário for válido, você encontrará um *.xctest* diretório dentro do *Plugins* diretório. Em nosso exemplo, o diretório é chamado*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)
   ```

   Para obter mais informações, consulte Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

```
We could not find the Info.plist file inside the .app directory. Please
unzip your test package and then open the .app directory, verify that the
Info.plist file is inside the directory, and try again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Se o pacote da XCTest interface do usuário for válido, você encontrará o *Info.plist* arquivo dentro do *.app* diretório. Em nosso exemplo abaixo, o diretório é chamado*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)
   ```

   Para obter mais informações, consulte Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING_IN_XCTEST_DIR

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

```
We could not find the Info.plist file inside the .xctest directory. Please
unzip your test package and then open the .xctest directory, verify that
the Info.plist file is inside the directory, and try again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip swift-sample-UI.ipa
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o pacote da XCTest interface do usuário for válido, você encontrará o *Info.plist* arquivo dentro do *.xctest* diretório. Em nosso exemplo abaixo, o diretório é chamado*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)
```

Para obter mais informações, consulte Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_CPU_ARCHITECTURE_VALUE_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

```
We could not the CPU architecture value in the Info.plist file. Please
unzip your test package and then open the Info.plist file inside the .app
directory, verify that the key "UIRequiredDeviceCapabilities" is specified,
and try again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip swift-sample-UI.ipa
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Você deve encontrar o *Info.plist* arquivo dentro de um *.app* diretório, como *swift-sampleUITests-Runner.app* no nosso exemplo:

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

3. Para encontrar o valor de arquitetura da CPU, você pode abrir o Info.plist usando o Xcode ou Python.

Para o Python, você pode instalar o módulo biplist executando o seguinte comando:

```
$ pip install biplist
```

4. Em seguida, abra o Python e execute o seguinte comando:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/
Info.plist')
print info_plist['UIRequiredDeviceCapabilities']
```

Um pacote de XCtest interface de usuário válido deve produzir uma saída como a seguinte:

```
['armv7']
```

Para obter mais informações, consulte Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_PLATFORM_VALUE_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

```
We could not find the platform value in the Info.plist. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the key "CFBundleSupportedPlatforms" is specified, and try again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Você deve encontrar o *Info.plist* arquivo dentro de um *.app* diretório, como *swift-sampleUITests-Runner.app* no nosso exemplo:

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

3. Para encontrar o valor da plataforma, você pode abrir o Info.plist usando o Xcode ou Python.

   Para o Python, você pode instalar o módulo biplist executando o seguinte comando:

```
$ pip install biplist
```

4. Em seguida, abra o Python e execute o seguinte comando:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['CFBundleSupportedPlatforms']
```

Um pacote de XCtest interface de usuário válido deve produzir uma saída como a seguinte:

```
['iPhoneOS']
```

Para obter mais informações, consulte Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_WRONG_PLATFORM_DEVICE_VALUE

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

```
We found the platform device value was wrong in the Info.plist file. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the value of the key "CFBundleSupportedPlatforms" does not contain the keyword "simulator", and try again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip swift-sample-UI.ipa
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Você deve encontrar o *`Info.plist`* arquivo dentro de um *`.app`* diretório, como *`swift-sampleUITests-Runner.app`* no nosso exemplo:

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

3. Para encontrar o valor da plataforma, você pode abrir o Info.plist usando o Xcode ou Python.

   Para o Python, você pode instalar o módulo biplist executando o seguinte comando:

   ```
   $ pip install biplist
   ```

4. Em seguida, abra o Python e execute o seguinte comando:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
   print info_plist['CFBundleSupportedPlatforms']
   ```

   Um pacote de XCtest interface de usuário válido deve produzir uma saída como a seguinte:

   ```
   ['iPhoneOS']
   ```

   Se o pacote de XCTest interface do usuário for válido, o valor não deverá conter a palavra-chave`simulator`.

   Para obter mais informações, consulte Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_FORM_FACTOR_VALUE_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

`We could not the form factor value in the Info.plist. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the key "UIDeviceFamily" is specified, and try again.`

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip swift-sample-UI.ipa
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

   Você deve encontrar o *Info.plist* arquivo dentro de um *.app* diretório, como *swift-sampleUITests-Runner.app* no nosso exemplo:

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

3. Para encontrar o valor do formato, você pode abrir o Info.plist usando o Xcode ou Python.

   Para o Python, você pode instalar o módulo biplist executando o seguinte comando:

```
$ pip install biplist
```

4. Em seguida, abra o Python e execute o seguinte comando:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['UIDeviceFamily']
```

   Um pacote de XCtest interface de usuário válido deve produzir uma saída como a seguinte:

```
[1, 2]
```

Para obter mais informações, consulte Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

```
We could not find the package name value in the Info.plist file. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the key "CFBundleIdentifier" is specified, and try again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Você deve encontrar o *Info.plist* arquivo dentro de um *.app* diretório, como *swift-sampleUITests-Runner.app* no nosso exemplo:

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

3. Para encontrar o valor do nome do pacote, você pode abrir o Info.plist usando o Xcode ou Python.

   Para o Python, você pode instalar o módulo biplist executando o seguinte comando:

```
$ pip install biplist
```

4. Em seguida, abra o Python e execute o seguinte comando:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['CFBundleIdentifier']
```

Um pacote de XCtest interface de usuário válido deve produzir uma saída como a seguinte:

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

Para obter mais informações, consulte Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

```
We could not find the executable value in the Info.plist file. Please
unzip your test package and then open the Info.plist file inside the .app
directory, verify that the key "CFBundleExecutable" is specified, and try
again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip swift-sample-UI.ipa
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Você deve encontrar o *Info.plist* arquivo dentro de um *.app* diretório, como *swift-sampleUITests-Runner.app* no nosso exemplo:

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

3. Para encontrar o valor do executável, você pode abrir o Info.plist usando o Xcode ou Python.

   Para o Python, você pode instalar o módulo biplist executando o seguinte comando:

   ```
   $ pip install biplist
   ```

4. Em seguida, abra o Python e execute o seguinte comando:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
   print info_plist['CFBundleExecutable']
   ```

   Um pacote de XCtest interface de usuário válido deve produzir uma saída como a seguinte:

   ```
   XCTRunner
   ```

   Para obter mais informações, consulte Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

```
We could not find the package name value in the Info.plist file inside
the .xctest directory. Please unzip your test package and then open
the Info.plist file inside the .xctest directory, verify that the key
"CFBundleIdentifier" is specified, and try again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip swift-sample-UI.ipa
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

   Você deve encontrar o *Info.plist* arquivo dentro de um *.app* diretório, como *swift-sampleUITests-Runner.app* no nosso exemplo:

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

3. Para encontrar o valor do nome do pacote, você pode abrir o Info.plist usando o Xcode ou Python.

   Para o Python, você pode instalar o módulo biplist executando o seguinte comando:

```
$ pip install biplist
```

4. Em seguida, abra o Python e execute o seguinte comando:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Plugins/
swift-sampleUITests.xctest/Info.plist')
print info_plist['CFBundleIdentifier']
```

   Um pacote de XCtest interface de usuário válido deve produzir uma saída como a seguinte:

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

Para obter mais informações, consulte Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_TEST_EXECUTABLE_VALUE_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

```
We could not find the executable value in the Info.plist file inside
the .xctest directory. Please unzip your test package and then open
the Info.plist file inside the .xctest directory, verify that the key
"CFBundleExecutable" is specified, and try again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Você deve encontrar o *Info.plist* arquivo dentro de um *.app* diretório, como *swift-sampleUITests-Runner.app* no nosso exemplo:

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

3. Para encontrar o valor do executável, você pode abrir o Info.plist usando o Xcode ou Python.

   Para o Python, você pode instalar o módulo biplist executando o seguinte comando:

```
$ pip install biplist
```

4. Em seguida, abra o Python e execute o seguinte comando:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Plugins/
swift-sampleUITests.xctest/Info.plist')
print info_plist['CFBundleExecutable']
```

Um pacote de XCtest interface de usuário válido deve produzir uma saída como a seguinte:

```
swift-sampleUITests
```

Para obter mais informações, consulte Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_MULTIPLE_APP_DIRS

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

`We found multiple .app directories inside your test package. Please unzip your test package, verify that only a single .app directory is present inside the package, then try again.`

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip swift-sample-UI.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o pacote de XCTest interface do usuário for válido, você deverá encontrar somente um único `.app` diretório, como `swift-sampleUITests-Runner.app` no nosso exemplo, dentro do pacote de teste.zip.

```
.
```

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

Para obter mais informações, consulte Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_MULTIPLE_IPA_DIRS

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

`We found multiple .ipa directories inside your test package. Please unzip your test package, verify that only a single .ipa directory is present inside the package, then try again.`

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swift-sample-UI.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Se o pacote de XCTest interface do usuário for válido, você deverá encontrar somente um único `.ipa` diretório, como `sampleUITests.ipa` no nosso exemplo, dentro do pacote de teste.zip.

   ```
   .
   `--swift-sample-UI.zip--(directory)
       `-- sampleUITests.ipa (directory)
               `-- Payload (directory)
                   `-- swift-sampleUITests-Runner.app (directory)
       `-- (any other files)
   ```

Para obter mais informações, consulte Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_BOTH_APP_E_IPA_DIR_PRESENT

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

```
We found both .app and .ipa files inside your test package. Please unzip
your test package, verify that only a single .app or .ipa file is present
inside the package, then try again.
```

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swift-sample-UI.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Se o pacote de XCTest interface do usuário for válido, você deverá encontrar um `.ipa` diretório semelhante `sampleUITests.ipa` ou um `.app` diretório como `swift-sampleUITests-Runner.app` em nosso exemplo dentro do pacote de teste.zip. Você pode consultar um exemplo de pacote de teste XCTEST_UI válido em nossa documentação sobre. Integrando a XCTest interface do usuário para iOS com o Device Farm

   ```
   .
   `--swift-sample-UI.zip--(directory)
       `-- sampleUITests.ipa (directory)
               `-- Payload (directory)
                   `-- swift-sampleUITests-Runner.app (directory)
      `-- (any other files)
   ```

   or

   ```
   .
   `--swift-sample-UI.zip--(directory)
       `-- swift-sampleUITests-Runner.app (directory)
   ```

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

Para obter mais informações, consulte Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_PRESENT_IN_ZIP

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

```
We found a Payload directory inside your .zip test package. Please unzip your test package, ensure that a Payload directory is not present in the package, then try again.
```

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swift-sample-UI.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Se o pacote de XCTest interface do usuário for válido, você não deverá encontrar um diretório de carga dentro do seu pacote de teste.

   ```
   .
   `--swift-sample-UI.zip--(directory)
       `-- swift-sampleUITests-Runner.app (directory)
               |-- Info.plist
               |-- Plugins (directory)
               `-- (any other files)
      `-- Payload (directory) [This directory should not be present]
               |-- (any other files)
      `-- (any other files)
   ```

Para obter mais informações, consulte Integrando a XCTest interface do usuário para iOS com o Device Farm.

Segurança em AWS Device Farm

A segurança na nuvem AWS é a maior prioridade. Como AWS cliente, você se beneficia de uma arquitetura de data center e rede criada para atender aos requisitos das organizações mais sensíveis à segurança.

A segurança é uma responsabilidade compartilhada entre você AWS e você. O modelo de responsabilidade compartilhada descreve isso como segurança da nuvem e segurança na nuvem:

- Segurança da nuvem — AWS é responsável por proteger a infraestrutura que executa AWS os serviços na AWS nuvem. AWS também fornece serviços que você pode usar com segurança. Auditores terceirizados testam e verificam regularmente a eficácia de nossa segurança como parte dos Programas de Conformidade Programas de AWS de . Para saber mais sobre os programas de conformidade que se aplicam AWS Device Farm, consulte Serviços da AWS no escopo do programa de conformidade .
- Segurança na nuvem: sua responsabilidade é determinada pelo serviço da AWS que você usa. Você também é responsável por outros fatores, incluindo a confidencialidade de seus dados, os requisitos da empresa e as leis e regulamentos aplicáveis.

Esta documentação o ajuda a entender como aplicar o modelo de responsabilidade compartilhada ao usar o Device Farm. Os tópicos a seguir mostram como configurar o Device Farm para atender aos seus objetivos de segurança e conformidade. Você também aprenderá a usar outros serviços da AWS que o ajudam a monitorar e proteger os recursos do Device Farm.

Tópicos

- Gerenciamento de identidade e acesso no AWS Device Farm
- Validação de conformidade para AWS Device Farm
- Proteção de dados em AWS Device Farm
- Resiliência em AWS Device Farm
- Segurança da infraestrutura em AWS Device Farm
- Análise e gerenciamento de vulnerabilidades de configuração no Device Farm
- Resposta a incidentes no Device Farm
- Registrar em log e monitorar no Device Farm
- Práticas recomendadas de segurança para o Device Farm

Gerenciamento de identidade e acesso no AWS Device Farm

Público

A forma como você usa AWS Identity and Access Management (IAM) difere, dependendo do trabalho que você faz no Device Farm.

Usuário do serviço: se você usa o serviço Device Farm para fazer seu trabalho, o administrador fornece as credenciais e as permissões necessárias. À medida que você usa mais recursos do Device Farm para fazer seu trabalho, pode precisar de permissões adicionais. Compreenda como o acesso é gerenciado pode ajudar a solicitar as permissões corretas ao administrador. Se não for possível acessar um recurso no Device Farm, consulte Solução de problemas de identidade e acesso ao AWS Device Farm.

Administrador de serviços: se você é responsável pelos recursos do Device Farm em sua empresa, provavelmente tem acesso total ao Device Farm. É seu trabalho determinar quais recursos e funcionalidades do Device Farm os usuários do serviço devem acessar. Envie as solicitações ao administrador do IAM para alterar as permissões dos usuários de serviço. Revise as informações nesta página para compreender os conceitos básicos do IAM. Para saber mais sobre como sua empresa pode usar o IAM com o Device Farm, consulte Como o AWS Device Farm funciona com o IAM.

Administrador do IAM: se você for um administrador do IAM, talvez queira saber detalhes sobre como escrever políticas para gerenciar o acesso ao Device Farm. Para ver exemplos de políticas baseadas em identidade do Device Farm que você pode usar no IAM, consulte Exemplos de políticas baseadas em identidade do AWS Device Farm.

Autenticar com identidades

A autenticação é como você faz login AWS usando suas credenciais de identidade. Você deve estar autenticado (conectado AWS) como usuário do Usuário raiz da conta da AWS IAM ou assumindo uma função do IAM.

Você pode entrar AWS como uma identidade federada usando credenciais fornecidas por meio de uma fonte de identidade. AWS IAM Identity Center Usuários (IAM Identity Center), a autenticação de login único da sua empresa e suas credenciais do Google ou do Facebook são exemplos de identidades federadas. Quando você faz login como identidade federada, o administrador já configurou anteriormente a federação de identidades usando perfis do IAM. Ao acessar AWS usando a federação, você está assumindo indiretamente uma função.

Dependendo do tipo de usuário que você é, você pode entrar no AWS Management Console ou no portal de AWS acesso. Para obter mais informações sobre como fazer login em AWS, consulte Como fazer login Conta da AWS no Guia do Início de Sessão da AWS usuário.

Se você acessar AWS programaticamente, AWS fornece um kit de desenvolvimento de software (SDK) e uma interface de linha de comando (CLI) para assinar criptograficamente suas solicitações usando suas credenciais. Se você não usa AWS ferramentas, você mesmo deve assinar as solicitações. Para obter mais informações sobre como usar o método recomendado para designar solicitações por conta própria, consulte Versão 4 do AWS Signature para solicitações de API no Guia do usuário do IAM.

Independente do método de autenticação usado, também pode ser necessário fornecer informações adicionais de segurança. Por exemplo, AWS recomenda que você use a autenticação multifator (MFA) para aumentar a segurança da sua conta. Para saber mais, consulte Autenticação multifator no Guia do usuário do AWS IAM Identity Center e Usar a autenticação multifator da AWS no IAM no Guia do usuário do IAM.

Conta da AWS usuário root

Ao criar uma Conta da AWS, você começa com uma identidade de login que tem acesso completo a todos Serviços da AWS os recursos da conta. Essa identidade é chamada de usuário Conta da AWS raiz e é acessada fazendo login com o endereço de e-mail e a senha que você usou para criar a conta. É altamente recomendável não usar o usuário-raiz para tarefas diárias. Proteja as credenciais do usuário-raiz e use-as para executar as tarefas que somente ele puder executar. Para obter a lista completa das tarefas que exigem login como usuário-raiz, consulte Tarefas que exigem credenciais de usuário-raiz no Guia do Usuário do IAM.

Grupos e usuários do IAM

Um usuário do IAM é uma identidade dentro da sua Conta da AWS que tem permissões específicas para uma única pessoa ou aplicativo. Sempre que possível, é recomendável contar com credenciais temporárias em vez de criar usuários do IAM com credenciais de longo prazo, como senhas e chaves de acesso. No entanto, se você tiver casos de uso específicos que exijam credenciais de longo prazo com usuários do IAM, é recomendável alternar as chaves de acesso. Para obter mais informações, consulte Alternar as chaves de acesso regularmente para casos de uso que exijam credenciais de longo prazo no Guia do Usuário do IAM.

Um grupo do IAM é uma identidade que especifica uma coleção de usuários do IAM. Não é possível fazer login como um grupo. É possível usar grupos para especificar permissões para vários usuários

de uma vez. Os grupos facilitam o gerenciamento de permissões para grandes conjuntos de usuários. Por exemplo, você pode ter um grupo chamado IAMAdminse conceder a esse grupo permissões para administrar recursos do IAM.

Usuários são diferentes de perfis. Um usuário é exclusivamente associado a uma pessoa ou a uma aplicação, mas um perfil pode ser assumido por qualquer pessoa que precisar dele. Os usuários têm credenciais permanentes de longo prazo, mas os perfis fornecem credenciais temporárias. Para saber mais, consulte Casos de uso para usuários do IAM no Guia do usuário do IAM.

Perfis do IAM

Uma função do IAM é uma identidade dentro da sua Conta da AWS que tem permissões específicas. Ele é semelhante a um usuário do IAM, mas não está associado a uma pessoa específica. Para assumir temporariamente uma função do IAM no AWS Management Console, você pode alternar de um usuário para uma função do IAM (console). Você pode assumir uma função chamando uma operação de AWS API AWS CLI ou usando uma URL personalizada. Para obter mais informações sobre métodos para usar perfis, consulte Métodos para assumir um perfil no Guia do usuário do IAM.

Perfis do IAM com credenciais temporárias são úteis nas seguintes situações:

- Acesso de usuário federado: para atribuir permissões a identidades federadas, é possível criar um perfil e definir permissões para ele. Quando uma identidade federada é autenticada, essa identidade é associada ao perfil e recebe as permissões definidas por ele. Para ter mais informações sobre perfis para federação, consulte Criar um perfil para um provedor de identidade de terceiros (federação) no Guia do usuário do IAM. Se usar o Centro de Identidade do IAM, configure um conjunto de permissões. Para controlar o que suas identidades podem acessar após a autenticação, o Centro de Identidade do IAM correlaciona o conjunto de permissões a um perfil no IAM. Para obter informações sobre conjuntos de permissões, consulte Conjuntos de Permissões no Guia do Usuário do AWS IAM Identity Center .
- Permissões temporárias para usuários do IAM: um usuário ou um perfil do IAM pode presumir um perfil do IAM para obter temporariamente permissões diferentes para uma tarefa específica.
- Acesso entre contas: é possível usar um perfil do IAM para permitir que alguém (uma entidade principal confiável) em outra conta acesse recursos em sua conta. Os perfis são a principal forma de conceder acesso entre contas. No entanto, com alguns Serviços da AWS, você pode anexar uma política diretamente a um recurso (em vez de usar uma função como proxy). Para conhecer a diferença entre perfis e políticas baseadas em recurso para acesso entre contas, consulte Acesso a recursos entre contas no IAM no Guia do usuário do IAM.

- Acesso entre serviços — Alguns Serviços da AWS usam recursos em outros Serviços da AWS. Por exemplo, quando você faz uma chamada em um serviço, é comum que esse serviço execute aplicativos na Amazon EC2 ou armazene objetos no Amazon S3. Um serviço pode fazer isso usando as permissões da entidade principal da chamada, usando um perfil de serviço ou um perfil vinculado ao serviço.
  - Sessões de acesso direto (FAS) — Quando você usa um usuário ou uma função do IAM para realizar ações AWS, você é considerado principal. Ao usar alguns serviços, você pode executar uma ação que inicia outra ação em um serviço diferente. O FAS usa as permissões do diretor chamando um AWS service (Serviço da AWS), combinadas com a solicitação AWS service (Serviço da AWS) para fazer solicitações aos serviços posteriores. As solicitações do FAS são feitas somente quando um serviço recebe uma solicitação que requer interações com outros Serviços da AWS ou com recursos para ser concluída. Nesse caso, você precisa ter permissões para executar ambas as ações. Para obter detalhes da política ao fazer solicitações de FAS, consulte Sessões de acesso direto.
  - Perfil de serviço: um perfil de serviço é um perfil do IAM que um serviço assume para executar ações em seu nome. Um administrador do IAM pode criar, modificar e excluir um perfil de serviço do IAM. Para obter mais informações, consulte Criar um perfil para delegar permissões a um AWS service (Serviço da AWS) no Guia do Usuário do IAM.
  - Função vinculada ao serviço — Uma função vinculada ao serviço é um tipo de função de serviço vinculada a um. AWS service (Serviço da AWS) O serviço pode presumir o perfil de executar uma ação em seu nome. As funções vinculadas ao serviço aparecem em você Conta da AWS e são de propriedade do serviço. Um administrador do IAM pode visualizar, mas não editar as permissões para perfis vinculados ao serviço.
- Aplicativos em execução na Amazon EC2 — Você pode usar uma função do IAM para gerenciar credenciais temporárias para aplicativos que estão sendo executados em uma EC2 instância e fazendo solicitações AWS CLI de AWS API. Isso é preferível a armazenar chaves de acesso na EC2 instância. Para atribuir uma AWS função a uma EC2 instância e disponibilizá-la para todos os aplicativos, você cria um perfil de instância anexado à instância. Um perfil de instância contém a função e permite que programas em execução na EC2 instância recebam credenciais temporárias. Para obter mais informações, consulte Usar uma função do IAM para conceder permissões a aplicativos executados em EC2 instâncias da Amazon no Guia do usuário do IAM.

Como o AWS Device Farm funciona com o IAM

Antes de usar o IAM para gerenciar o acesso ao Device Farm, você deve entender quais recursos do IAM estão disponíveis para uso com o Device Farm. Para ter uma visão geral de como o Device Farm e outros AWS serviços funcionam com o IAM, consulte AWS Services That Work with IAM no Guia do usuário do IAM.

Tópicos

- Políticas baseadas em identidade do Device Farm
- Políticas baseadas em recursos do Device Farm
- Listas de controle de acesso
- Autorização baseada em tags do Device Farm
- Perfis do IAM do Device Farm

Políticas baseadas em identidade do Device Farm

Com as políticas baseadas em identidade do IAM, é possível especificar ações ou recursos permitidos ou negados, além das condições sob as quais as ações são permitidas ou negadas. O Device Farm oferece suporte a ações, recursos e chaves de condição específicos. Para conhecer todos os elementos usados em uma política JSON, consulte Referência de elementos de política JSON do IAM no Guia do usuário do IAM.

Ações

Os administradores podem usar políticas AWS JSON para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento `Action` de uma política JSON descreve as ações que podem ser usadas para permitir ou negar acesso em uma política. As ações de política geralmente têm o mesmo nome da operação de AWS API associada. Existem algumas exceções, como ações somente de permissão, que não têm uma operação de API correspondente. Algumas operações também exigem várias ações em uma política. Essas ações adicionais são chamadas de ações dependentes.

Incluem ações em uma política para conceder permissões para executar a operação associada.

As ações de política no Device Farm usam o seguinte prefixo antes da ação: `devicefarm:`. Por exemplo, para conceder permissão a alguém para iniciar sessões do Selenium com a operação de API `CreateTestGridUrl` de teste de navegador de desktop do Device Farm, você inclui a

ação `devicefarm:CreateTestGridUrl` na política. As instruções de política devem incluir um elemento `Action` ou `NotAction`. O Device Farm define seu próprio conjunto de ações que descrevem as tarefas que você pode executar com esse serviço.

Para especificar várias ações em uma única instrução, separe-as com vírgulas, como segue:

```
"Action": [
      "devicefarm:action1",
      "devicefarm:action2"
```

Você também pode especificar várias ações usando caracteres curinga (*). Por exemplo, para especificar todas as ações que começam com a palavra `List`, inclua a seguinte ação:

```
"Action": "devicefarm:List*"
```

Para ver uma lista de ações do Device Farm, consulte Actions defined by AWS Device Farm na Referência de autorização de serviço do IAM.

Recursos

Os administradores podem usar políticas AWS JSON para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento de política JSON `Resource` especifica o objeto ou os objetos aos quais a ação se aplica. As instruções devem incluir um elemento `Resource` ou `NotResource`. Como prática recomendada, especifique um recurso usando seu nome do recurso da Amazon (ARN). Isso pode ser feito para ações que oferecem compatibilidade com um tipo de recurso específico, conhecido como permissões em nível de recurso.

Para ações que não oferecem compatibilidade com permissões em nível de recurso, como operações de listagem, use um curinga (*) para indicar que a instrução se aplica a todos os recursos.

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

O recurso de EC2 instância da Amazon tem o seguinte ARN:

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

Para obter mais informações sobre o formato de ARNs, consulte Amazon Resource Names (ARNs) e AWS Service Namespaces.

Por exemplo, para especificar a instância `i-1234567890abcdef0` na instrução, use o seguinte ARN:

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:instance/i-1234567890abcdef0"
```

Para especificar todas as instâncias que pertencem a uma conta, use o caractere curinga (*):

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:instance/*"
```

Algumas ações do Device Farm, como as de criação de recursos, não podem ser executadas em um recurso. Nesses casos, você deve utilizar o caractere curinga (*).

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

Muitas ações de EC2 API da Amazon envolvem vários recursos. Por exemplo, `AttachVolume` anexa um volume do Amazon EBS a uma instância, portanto, um usuário do IAM deve ter permissões para usar o volume e a instância. Para especificar vários recursos em uma única instrução, separe-os ARNs com vírgulas.

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

Para ver uma lista dos tipos de recursos do Device Farm e seus ARNs, consulte Tipos de recursos definidos AWS Device Farm na Referência de autorização de serviço do IAM. Para saber com quais ações você pode especificar o ARN de cada recurso, consulte Actions defined by AWS Device Farm na Referência de autorização de serviço do IAM.

Chaves de condição

Os administradores podem usar políticas AWS JSON para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento `Condition` (ou bloco `Condition`) permite que você especifique condições nas quais uma instrução estiver em vigor. O elemento `Condition` é opcional. É possível criar expressões condicionais que usem agentes de condição, como “igual a” ou “menor que”, para fazer a condição da política corresponder aos valores na solicitação.

Se você especificar vários elementos de `Condition` em uma declaração ou várias chaves em um único elemento de `Condition`, a AWS os avaliará usando uma operação lógica `AND`. Se você especificar vários valores para uma única chave de condição, AWS avalia a condição usando uma `OR` operação lógica. Todas as condições devem ser atendidas antes que as permissões da instrução sejam concedidas.

Você também pode usar variáveis de espaço reservado ao especificar condições. Por exemplo, é possível conceder a um usuário do IAM permissão para acessar um recurso somente se ele estiver marcado com seu nome de usuário do IAM. Para obter mais informações, consulte Elementos da política do IAM: variáveis e tags no Guia do usuário do IAM.

AWS suporta chaves de condição globais e chaves de condição específicas do serviço. Para ver todas as chaves de condição AWS globais, consulte as chaves de contexto de condição AWS global no Guia do usuário do IAM.

O Device Farm define seu próprio conjunto de chaves de condição e também permite o uso de algumas chaves de condição globais. Para ver todas as chaves de condição AWS globais, consulte Chaves de contexto de condição AWS global no Guia do usuário do IAM.

Para ver uma lista das chaves de condição do Device Farm, consulte Condition keys for AWS Device Farm na Referência de autorização de serviço do IAM. Para saber com quais ações e recursos você pode usar uma chave de condição, consulte Actions defined by AWS Device Farm na Referência de autorização de serviço do IAM.

Exemplos

Para ver exemplos de políticas baseadas em identidade do Device Farm, consulte Exemplos de políticas baseadas em identidade do AWS Device Farm.

Políticas baseadas em recursos do Device Farm

O Device Farm não é compatível com políticas baseadas em recursos.

Listas de controle de acesso

O Device Farm não suporta listas de controle de acesso (ACLs).

Autorização baseada em tags do Device Farm

Você pode anexar tags aos recursos do Device Farm ou passar tags em uma solicitação para o Device Farm. Para controlar o acesso baseado em tags, forneça informações sobre as

tags no elemento de condição de uma política usando as `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` ou chaves de condição `aws:TagKeys`. Para obter mais informações sobre a marcação de recursos do Device Farm, consulte Marcação no Device Farm.

Para visualizar um exemplo de política baseada em identidade para limitar o acesso a um recurso baseado em tags desse recurso, consulte Visualizando projetos de teste do navegador de desktop Device Farm com base em tags.

Perfis do IAM do Device Farm

Uma função do IAM é uma entidade na sua AWS conta que tem permissões específicas.

Uso de credenciais temporárias com o Device Farm

O Device Farm é compatível com o uso de credenciais temporárias.

Você pode usar credenciais temporárias para fazer login com a federação e assumir um perfil do IAM ou um perfil entre contas. Você obtém credenciais de segurança temporárias chamando operações de AWS STS API, como AssumeRoleou GetFederationToken.

Perfis vinculados ao serviço

As funções vinculadas ao serviço permitem que AWS os serviços acessem recursos em outros serviços para concluir uma ação em seu nome. Os perfis vinculados a serviço aparecem em sua conta do IAM e são de propriedade do serviço. Um administrador do IAM pode visualizar, mas não pode editar, as permissões das funções vinculadas a serviços.

O Device Farm usa perfis vinculados ao serviço no recurso de teste do navegador de desktop do Device Farm. Para obter informações sobre esses perfis, consulte Using Service-Linked Roles in Device Farm desktop browser testing no guia do desenvolvedor.

Perfis de serviço

O Device Farm não é compatível com perfis de serviço.

Esse atributo permite que um serviço assuma um perfil de serviço em seu nome. O perfil permite que o serviço acesse recursos em outros serviços para concluir uma ação em seu nome. Os perfis de serviço aparecem em sua conta do IAM e são de propriedade da conta. Isso significa que um administrador do IAM pode alterar as permissões para esse perfil. Porém, fazer isso pode alterar a funcionalidade do serviço.

Gerenciar o acesso usando políticas

Você controla o acesso AWS criando políticas e anexando-as a AWS identidades ou recursos. Uma política é um objeto AWS que, quando associada a uma identidade ou recurso, define suas permissões. AWS avalia essas políticas quando um principal (usuário, usuário raiz ou sessão de função) faz uma solicitação. As permissões nas políticas determinam se a solicitação será permitida ou negada. A maioria das políticas é armazenada AWS como documentos JSON. Para obter mais informações sobre a estrutura e o conteúdo de documentos de políticas JSON, consulte Visão geral das políticas JSON no Guia do usuário do IAM.

Os administradores podem usar políticas AWS JSON para especificar quem tem acesso ao quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

Por padrão, usuários e perfis não têm permissões. Para conceder permissão aos usuários para executar ações nos recursos que eles precisam, um administrador do IAM pode criar políticas do IAM. O administrador pode então adicionar as políticas do IAM aos perfis e os usuários podem assumir os perfis.

As políticas do IAM definem permissões para uma ação independentemente do método usado para executar a operação. Por exemplo, suponha que você tenha uma política que permite a ação `iam:GetRole`. Um usuário com essa política pode obter informações de função da AWS Management Console AWS CLI, da ou da AWS API.

Políticas baseadas em identidade

As políticas baseadas em identidade são documentos de políticas de permissões JSON que você pode anexar a uma identidade, como usuário, grupo de usuários ou perfil do IAM. Essas políticas controlam quais ações os usuários e perfis podem realizar, em quais recursos e em que condições. Para saber como criar uma política baseada em identidade, consulte Definir permissões personalizadas do IAM com as políticas gerenciadas pelo cliente no Guia do Usuário do IAM.

As políticas baseadas em identidade podem ser categorizadas como políticas em linha ou políticas gerenciadas. As políticas em linha são anexadas diretamente a um único usuário, grupo ou perfil. As políticas gerenciadas são políticas autônomas que você pode associar a vários usuários, grupos e funções em seu Conta da AWS. As políticas AWS gerenciadas incluem políticas gerenciadas e políticas gerenciadas pelo cliente. Para saber como escolher entre uma política gerenciada ou uma política em linha, consulte Escolher entre políticas gerenciadas e políticas em linha no Guia do usuário do IAM.

A tabela a seguir descreve as políticas gerenciadas pela AWS do Device Farm.

Alteração	Descrição	Data
AWSDeviceFarmFullAccess	Fornece acesso total a todas as operações do AWS Device Farm.	15 de julho de 2015
AWSServiceRoleForDeviceFarmTestGrid	Permite que o Device Farm acesse recursos da AWS em seu nome.	20 de maio de 2021

Outros tipos de política

AWS oferece suporte a tipos de políticas adicionais menos comuns. Esses tipos de política podem definir o máximo de permissões concedidas a você pelos tipos de política mais comuns.

- Limites de permissões: um limite de permissões é um recurso avançado no qual você define o máximo de permissões que uma política baseada em identidade pode conceder a uma entidade do IAM (usuário ou perfil do IAM). É possível definir um limite de permissões para uma entidade. As permissões resultantes são a interseção das políticas baseadas em identidade de uma entidade com seus limites de permissões. As políticas baseadas em recurso que especificam o usuário ou o perfil no campo `Principal` não são limitadas pelo limite de permissões. Uma negação explícita em qualquer uma dessas políticas substitui a permissão. Para obter mais informações sobre limites de permissões, consulte Limites de permissões para identidades do IAM no Guia do usuário do IAM.
- Políticas de controle de serviço (SCPs) — SCPs são políticas JSON que especificam as permissões máximas para uma organização ou unidade organizacional (OU) em AWS Organizations. AWS Organizations é um serviço para agrupar e gerenciar centralmente vários Contas da AWS que sua empresa possui. Se você habilitar todos os recursos em uma organização, poderá aplicar políticas de controle de serviço (SCPs) a qualquer uma ou a todas as suas contas. O SCP limita as permissões para entidades nas contas dos membros, incluindo cada uma Usuário raiz da conta da AWS. Para obter mais informações sobre Organizations e SCPs, consulte Políticas de controle de serviços no Guia AWS Organizations do Usuário.
- Políticas de controle de recursos (RCPs) — RCPs são políticas JSON que você pode usar para definir o máximo de permissões disponíveis para recursos em suas contas sem atualizar as políticas do IAM anexadas a cada recurso que você possui. O RCP limita as permissões para recursos nas contas dos membros e pode afetar as permissões efetivas para identidades,

incluindo a Usuário raiz da conta da AWS, independentemente de pertencerem à sua organização. Para obter mais informações sobre Organizations e RCPs, incluindo uma lista Serviços da AWS desse suporte RCPs, consulte Políticas de controle de recursos (RCPs) no Guia AWS Organizations do usuário.

- Políticas de sessão: são políticas avançadas que você transmite como um parâmetro quando cria de forma programática uma sessão temporária para um perfil ou um usuário federado. As permissões da sessão resultante são a interseção das políticas baseadas em identidade do usuário ou do perfil e das políticas de sessão. As permissões também podem ser provenientes de uma política baseada em recursos. Uma negação explícita em qualquer uma dessas políticas substitui a permissão. Para obter mais informações, consulte Políticas de sessão no Guia do usuário do IAM.

Vários tipos de política

Quando vários tipos de política são aplicáveis a uma solicitação, é mais complicado compreender as permissões resultantes. Para saber como AWS determinar se uma solicitação deve ser permitida quando vários tipos de políticas estão envolvidos, consulte Lógica de avaliação de políticas no Guia do usuário do IAM.

Exemplos de políticas baseadas em identidade do AWS Device Farm

Por padrão, os perfis e usuários do IAM não têm permissão para criar ou modificar recursos do Device Farm. Eles também não podem realizar tarefas usando a AWS API AWS Management Console AWS CLI, ou. Um administrador do IAM deve criar políticas do IAM que concedam aos usuários e perfis permissão para executarem operações de API específicas nos recursos especificados de que precisam. O administrador deve anexar essas políticas aos usuários ou grupos do IAM que exigem essas permissões.

Para saber como criar uma política baseada em identidade do IAM usando esses exemplos de documentos de política JSON, consulte Criar políticas na guia JSON no Guia do usuário do IAM.

Tópicos

- Práticas recomendadas de política
- Permitir que os usuários visualizem suas próprias permissões
- Acesso a um projeto de teste de navegador de desktop do Device Farm
- Visualizando projetos de teste do navegador de desktop Device Farm com base em tags

Práticas recomendadas de política

As políticas baseadas em identidade determinam se alguém pode criar, acessar ou excluir recursos do Device Farm em sua conta. Essas ações podem incorrer em custos para sua Conta da AWS. Ao criar ou editar políticas baseadas em identidade, siga estas diretrizes e recomendações:

- Comece com as políticas AWS gerenciadas e avance para as permissões de privilégios mínimos — Para começar a conceder permissões aos seus usuários e cargas de trabalho, use as políticas AWS gerenciadas que concedem permissões para muitos casos de uso comuns. Eles estão disponíveis no seu Conta da AWS. Recomendamos que você reduza ainda mais as permissões definindo políticas gerenciadas pelo AWS cliente que sejam específicas para seus casos de uso. Para obter mais informações, consulte Políticas gerenciadas pela AWS ou Políticas gerenciadas pela AWS para funções de trabalho no Guia do usuário do IAM.
- Aplique permissões de privilégio mínimo: ao definir permissões com as políticas do IAM, conceda apenas as permissões necessárias para executar uma tarefa. Você faz isso definindo as ações que podem ser executadas em recursos específicos sob condições específicas, também conhecidas como permissões de privilégio mínimo. Para obter mais informações sobre como usar o IAM para aplicar permissões, consulte Políticas e permissões no IAM no Guia do usuário do IAM.
- Use condições nas políticas do IAM para restringir ainda mais o acesso: você pode adicionar uma condição às políticas para limitar o acesso a ações e recursos. Por exemplo, você pode escrever uma condição de política para especificar que todas as solicitações devem ser enviadas usando SSL. Você também pode usar condições para conceder acesso às ações de serviço se elas forem usadas por meio de uma ação específica AWS service (Serviço da AWS), como AWS CloudFormation. Para obter mais informações, consulte Elementos da política JSON do IAM: condição no Guia do usuário do IAM.
- Use o IAM Access Analyzer para validar suas políticas do IAM a fim de garantir permissões seguras e funcionais: o IAM Access Analyzer valida as políticas novas e existentes para que elas sigam a linguagem de política do IAM (JSON) e as práticas recomendadas do IAM. O IAM Access Analyzer oferece mais de cem verificações de política e recomendações práticas para ajudar a criar políticas seguras e funcionais. Para obter mais informações, consulte Validação de políticas do IAM Access Analyzer no Guia do Usuário do IAM.
- Exigir autenticação multifator (MFA) — Se você tiver um cenário que exija usuários do IAM ou um usuário root, ative Conta da AWS a MFA para obter segurança adicional. Para exigir MFA quando as operações de API forem chamadas, adicione condições de MFA às suas políticas. Para obter mais informações, consulte Configuração de acesso à API protegido por MFA no Guia do Usuário do IAM.

Para obter mais informações sobre as práticas recomendadas do IAM, consulte Práticas recomendadas de segurança no IAM no Guia do usuário do IAM.

Permitir que os usuários visualizem suas próprias permissões

Este exemplo mostra como criar uma política que permita que os usuários do IAM visualizem as políticas gerenciadas e em linha anexadas a sua identidade de usuário. Essa política inclui permissões para concluir essa ação no console ou programaticamente usando a API AWS CLI ou AWS .

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Acesso a um projeto de teste de navegador de desktop do Device Farm

Neste exemplo, você deseja conceder a um usuário do IAM em sua AWS conta acesso a um dos seus projetos de teste do navegador de desktop Device Farm,. `arn:aws:devicefarm:us-west-2:111122223333:testgrid-project:123e4567-e89b-12d3-a456-426655441111` Você deseja que a conta seja capaz de ver itens relacionados ao projeto.

Além do endpoint `devicefarm:GetTestGridProject`, a conta deve ter os endpoints `devicefarm:ListTestGridSessionArtifacts` `devicefarm:ListTestGridSessions`, `devicefarm:GetTestGridSession` e `devicefarm:ListTestGridSessionActions`.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"GetTestGridProject",
         "Effect":"Allow",
         "Action":[
            "devicefarm:GetTestGridProject"
         ],
         "Resource":"arn:aws:devicefarm:us-west-2:111122223333:testgrid-
project:123e4567-e89b-12d3-a456-426655441111"
      },
      {
         "Sid":"ViewProjectInfo",
         "Effect":"Allow",
         "Action":[
            "devicefarm:ListTestGridSessions",
            "devicefarm:ListTestGridSessionActions",
            "devicefarm:ListTestGridSessionArtifacts"
         ],
         "Resource":"arn:aws:devicefarm:us-west-2:111122223333:testgrid-*:123e4567-
e89b-12d3-a456-426655441111/*"
      }
   ]
}
```

Se estiver usando sistemas de CI, você deverá fornecer a cada executor de CI credenciais de acesso exclusivas. Por exemplo, é improvável que um sistema de CI precise de mais permissões do que `devicefarm:ScheduleRun` ou `devicefarm:CreateUpload`. A política de IAM a seguir descreve uma política mínima para permitir que um executor de CI inicie um teste de um novo

aplicativo nativo do Device Farm criando um upload e usando-o para agendar uma execução de teste:

```
{
   "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/*",
            ]
         }
    ]
}
```

Visualizando projetos de teste do navegador de desktop Device Farm com base em tags

Você pode usar condições em sua política baseada em identidade para controlar o acesso aos recursos do Device Farm com base em tags. Este exemplo mostra como você pode criar uma política que permite a visualização de projetos e sessões. A permissão será concedida se a tag `0wner` do recurso solicitado corresponder ao nome de usuário da conta solicitante.

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

É possível anexar essa política aos usuários do IAM na sua conta. Se um usuário chamado `richard-roe` tentar visualizar um projeto ou uma sessão do Device Farm, o projeto deverá ser marcado com `Owner=richard-roe` ou `owner=richard-roe`. Caso contrário, o usuário terá o acesso negado. A chave da tag de condição `Owner` corresponde a `Owner` e a `owner` porque os nomes de chaves de condição não diferenciam letras maiúsculas de minúsculas. Para obter mais informações, consulte IAM JSON Policy Elements: Condition (Elementos da política JSON do IAM: Condição) no Guia do usuário do IAM.

Solução de problemas de identidade e acesso ao AWS Device Farm

Use as informações a seguir para ajudá-lo a diagnosticar e corrigir problemas comuns que você pode encontrar ao trabalhar com o Device Farm e o IAM.

Não estou autorizado a realizar uma ação no Device Farm

Se você receber uma mensagem de erro AWS Management Console informando que você não está autorizado a realizar uma ação, entre em contato com o administrador para obter ajuda. O administrador é a pessoa que forneceu a você o seu nome de usuário e senha.

O exemplo de erro a seguir ocorre quando o usuário do IAM, `mateojackson`, tenta usar o console para visualizar detalhes sobre uma execução, mas não tem permissões `devicefarm:GetRun`.

```
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
```

Nesse caso, Mateo pede ao administrador para atualizar suas políticas para que ele tenha acesso ao `devicefarm:GetRun` no recurso `arn:aws:devicefarm:us-west-2:123456789101:run:123e4567-e89b-12d3-a456-426655440000/123e4567-e89b-12d3-a456-426655441111` usando a ação `devicefarm:GetRun`.

Não estou autorizado a realizar iam: PassRole

Se você receber um erro informando que não está autorizado a executar a ação `iam:PassRole`, suas políticas deverão ser atualizadas para permitir que você passe um perfil para o Device Farm.

Alguns Serviços da AWS permitem que você passe uma função existente para esse serviço em vez de criar uma nova função de serviço ou uma função vinculada ao serviço. Para fazer isso, é preciso ter permissões para passar o perfil para o serviço.

O exemplo de erro a seguir ocorre quando um usuário do IAM chamado `marymajor` tenta usar o console para executar uma ação no Device Farm. No entanto, a ação exige que o serviço tenha permissões concedidas por um perfil de serviço. Mary não tem permissões para passar o perfil para o serviço.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Nesse caso, as políticas de Mary devem ser atualizadas para permitir que ela realize a ação `iam:PassRole`.

Se precisar de ajuda, entre em contato com seu AWS administrador. Seu administrador é a pessoa que forneceu suas credenciais de login.

Quero visualizar minhas chaves de acesso

Depois de criar suas chaves de acesso de usuário do IAM, é possível visualizar seu ID da chave de acesso a qualquer momento. No entanto, você não pode visualizar sua chave de acesso secreta novamente. Se você perder sua chave secreta, crie um novo par de chaves de acesso.

As chaves de acesso consistem em duas partes: um ID de chave de acesso (por exemplo, `AKIAIOSFODNN7EXAMPLE`) e uma chave de acesso secreta (por exemplo, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Como um nome de usuário e uma senha, você deve usar o ID da chave de acesso e a chave de acesso secreta em conjunto para autenticar suas solicitações. Gerencie suas chaves de acesso de forma tão segura quanto você gerencia seu nome de usuário e sua senha.

> **Important**
>
> Não forneça as chaves de acesso a terceiros, mesmo que seja para ajudar a encontrar o ID de usuário canônico. Ao fazer isso, você pode dar a alguém acesso permanente ao seu Conta da AWS.

Ao criar um par de chaves de acesso, você é solicitado a guardar o ID da chave de acesso e a chave de acesso secreta em um local seguro. A chave de acesso secreta só está disponível no momento em que é criada. Se você perder sua chave de acesso secreta, será necessário adicionar novas chaves de acesso para seu usuário do IAM. Você pode ter no máximo duas chaves de acesso. Se você já tiver duas, você deverá excluir um par de chaves para poder criar um novo. Para visualizar as instruções, consulte Gerenciar chaves de acesso no Guia do usuário do IAM.

Sou um administrador e quero permitir que outras pessoas acessem o Device Farm

Para permitir que outras pessoas acessem o Device Farm, você deve conceder permissão às pessoas ou aplicativos que precisam de acesso. Se você estiver usando o AWS IAM Identity Center para gerenciar pessoas e aplicações, atribua conjuntos de permissões a usuários ou grupos para definir o nível de acesso. Os conjuntos de permissões criam e atribuem automaticamente políticas do IAM aos perfis do IAM associados à pessoa ou aplicação. Para ter mais informações, consulte Conjuntos de permissões no Guia do usuário do AWS IAM Identity Center .

Se você não estiver usando o Centro de Identidade do IAM, deverá criar entidades do IAM (usuários ou perfis) para as pessoas ou aplicações que precisam de acesso. Em seguida, você deve anexar uma política à entidade que concede a ela as permissões corretas no Device Farm. Depois que as permissões forem concedidas, forneça as credenciais ao usuário ou desenvolvedor da aplicação. Eles usarão essas credenciais para acessar AWS. Para saber mais sobre como criar grupos, políticas, permissões e usuários do IAM, consulte Identidades do IAM e Políticas e permissões no IAM no Guia do usuário do IAM.

Quero permitir que pessoas fora da minha conta da AWS acessem meus recursos do Device Farm

É possível criar um perfil que os usuários de outras contas ou pessoas fora da sua organização podem usar para acessar seus recursos. É possível especificar quem é confiável para assumir o perfil. Para serviços que oferecem suporte a políticas baseadas em recursos ou listas de controle de acesso (ACLs), você pode usar essas políticas para conceder às pessoas acesso aos seus recursos.

Para saber mais, consulte:

- Para saber se o Device Farm é compatível com esses recursos, consulte Como o AWS Device Farm funciona com o IAM.
- Para saber como fornecer acesso aos seus recursos em todos os Contas da AWS que você possui, consulte Como fornecer acesso a um usuário do IAM em outro Conta da AWS que você possui no Guia do usuário do IAM.
- Para saber como fornecer acesso aos seus recursos a terceiros Contas da AWS, consulte Como fornecer acesso Contas da AWS a terceiros no Guia do usuário do IAM.
- Para saber como conceder acesso por meio da federação de identidades, consulte Conceder acesso a usuários autenticados externamente (federação de identidades) no Guia do usuário do IAM.
- Para conhecer a diferença entre perfis e políticas baseadas em recurso para acesso entre contas, consulte Acesso a recursos entre contas no IAM no Guia do usuário do IAM.

Validação de conformidade para AWS Device Farm

Auditores terceirizados avaliam a segurança e a conformidade AWS Device Farm como parte de vários programas de AWS conformidade. Isso inclui SOC, PCI, FedRAMP, HIPAA e outros. AWS Device Farm não está no escopo de nenhum programa de AWS conformidade.

Para obter uma lista de AWS serviços no escopo de programas de conformidade específicos, consulte Serviços da AWS no escopo do programa de conformidade . Para obter informações gerais, consulte Programas de AWS conformidade Programas AWS de .

Você pode baixar relatórios de auditoria de terceiros usando AWS Artifact. Para obter mais informações, consulte Fazer download de relatórios no AWS Artifact.

Sua responsabilidade de conformidade ao usar o Device Farm é determinada pela confidencialidade de seus dados, pelos objetivos de conformidade da sua empresa e pelas leis e regulamentações aplicáveis. AWS fornece os seguintes recursos para ajudar na conformidade:

- Guias de início rápido de segurança e compatibilidade: esses guias de implantação abordam as considerações de arquitetura e fornecem etapas para implantação de ambientes de linha de base focados em compatibilidade e segurança na AWS.
- AWS Recursos de https://aws.amazon.com/compliance/resources/ de conformidade — Essa coleção de pastas de trabalho e guias pode ser aplicada ao seu setor e local.

- Avaliação de recursos com regras no Guia do AWS Config desenvolvedor — AWS Config avalia o quão bem suas configurações de recursos estão em conformidade com as práticas internas, as diretrizes do setor e os regulamentos.
- AWS Security Hub— Esse AWS serviço fornece uma visão abrangente do seu estado de segurança interno, AWS que ajuda você a verificar sua conformidade com os padrões e as melhores práticas do setor de segurança.

Proteção de dados em AWS Device Farm

O modelo de responsabilidade AWS compartilhada modelo de se aplica à proteção de dados em AWS Device Farm (Device Farm). Conforme descrito neste modelo, AWS é responsável por proteger a infraestrutura global que executa todos os Nuvem AWS. Você é responsável por manter o controle sobre o conteúdo hospedado nessa infraestrutura. Você também é responsável pelas tarefas de configuração e gerenciamento de segurança dos Serviços da AWS que usa. Para obter mais informações sobre a privacidade de dados, consulte as Data Privacy FAQ. Para obter mais informações sobre a proteção de dados na Europa, consulte a postagem do blog AWS Shared Responsibility Model and RGPD no Blog de segurança da AWS .

Para fins de proteção de dados, recomendamos que você proteja Conta da AWS as credenciais e configure usuários individuais com AWS IAM Identity Center ou AWS Identity and Access Management (IAM). Dessa maneira, cada usuário receberá apenas as permissões necessárias para cumprir suas obrigações de trabalho. Recomendamos também que você proteja seus dados das seguintes formas:

- Use uma autenticação multifator (MFA) com cada conta.
- Use SSL/TLS para se comunicar com os recursos. AWS Exigimos TLS 1.2 e recomendamos TLS 1.3.
- Configure a API e o registro de atividades do usuário com AWS CloudTrail. Para obter informações sobre o uso de CloudTrail trilhas para capturar AWS atividades, consulte Como trabalhar com CloudTrail trilhas no Guia AWS CloudTrail do usuário.
- Use soluções de AWS criptografia, juntamente com todos os controles de segurança padrão Serviços da AWS.
- Use serviços gerenciados de segurança avançada, como o Amazon Macie, que ajuda a localizar e proteger dados sigilosos armazenados no Amazon S3.
- Se você precisar de módulos criptográficos validados pelo FIPS 140-3 ao acessar AWS por meio de uma interface de linha de comando ou de uma API, use um endpoint FIPS. Para obter

mais informações sobre os endpoints FIPS disponíveis, consulte Federal Information Processing Standard (FIPS) 140-3.

É altamente recomendável que nunca sejam colocadas informações confidenciais ou sigilosas, como endereços de e-mail de clientes, em tags ou campos de formato livre, como um campo Nome. Isso inclui quando você trabalha com o Device Farm ou outro Serviços da AWS usando o console AWS CLI, a API ou AWS SDKs. Quaisquer dados inseridos em tags ou em campos de texto de formato livre usados para nomes podem ser usados para logs de faturamento ou de diagnóstico. Se você fornecer um URL para um servidor externo, recomendemos fortemente que não sejam incluídas informações de credenciais no URL para validar a solicitação a esse servidor.

Criptografia em trânsito

Os endpoints do Device Farm suportam somente HTTPS assinado (SSL/TLS) requests except where otherwise noted. All content retrieved from or placed in Amazon S3 through upload URLs is encrypted using SSL/TLS. Para obter mais informações sobre como as solicitações HTTPS são registradas AWS, consulte Assinatura de solicitações da AWS API na Referência AWS geral.

É sua responsabilidade criptografar e proteger qualquer comunicação que seus aplicativos testados façam e quaisquer aplicativos extras instalados no processo de execução de testes no dispositivo.

Criptografia em repouso

O recurso de teste do navegador de desktop do Device Farm oferece suporte à criptografia em repouso para artefatos gerados durante os testes.

Os dados de teste do dispositivo móvel físico do Device Farm não são criptografados em repouso.

Retenção de dados

Os dados no Device Farm são retidos por um tempo limitado. Depois que o período de retenção expirar, os dados serão removidos do armazenamento de backup do Device Farm.

Tipo de conteúdo	Período de retenção (dias)	Período de retenção de metadados (dias)
Aplicativos carregados	30	30
Pacotes de teste carregados	30	30

Tipo de conteúdo	Período de retenção (dias)	Período de retenção de metadados (dias)
Logs	400	400
Gravações de vídeo e outros artefatos	400	400

É sua responsabilidade arquivar qualquer conteúdo que você queira reter por períodos mais longos.

Gerenciamento de dados

Os dados no Device Farm são gerenciados de forma diferente, dependendo de quais recursos são usados. Esta seção explica como os dados são gerenciados durante e após o uso do Device Farm.

Teste do navegador de desktop

As instâncias usadas durante as sessões do Selenium não são salvas. Todos os dados gerados como resultado de interações do navegador são descartados quando a sessão termina.

Atualmente, esse recurso oferece suporte à criptografia em repouso para artefatos gerados durante o teste.

Teste de dispositivo físico

As seções a seguir fornecem informações sobre as etapas AWS necessárias para limpar ou destruir dispositivos depois de usar o Device Farm.

Os dados de teste do dispositivo móvel físico do Device Farm não são criptografados em repouso.

Frotas de dispositivo público

Após a conclusão da execução do teste, o Device Farm realiza uma série de tarefas de limpeza em cada dispositivo da frota pública de dispositivos, incluindo a desinstalação do seu aplicativo. Se não conseguirmos verificar a desinstalação do aplicativo ou qualquer uma das outras etapas de limpeza, o dispositivo receberá uma redefinição de fábrica antes de ser recolocado em uso.

> **Note**
>
> Em alguns casos, é possível que os dados persistam entre as sessões, especialmente se você usar o sistema do dispositivo fora do contexto do seu aplicativo. Por esse motivo, e

como o Device Farm captura vídeos e logs de atividades que ocorrem durante o uso de cada dispositivo, recomendamos que você não insira informações confidenciais (por exemplo, conta do Google ou ID da Apple), informações pessoais e outros detalhes sensíveis à segurança durante o teste automatizado e as sessões de acesso remoto.

Dispositivos privados

Após a expiração ou o encerramento do contrato de dispositivos privados, o dispositivo é removido do uso e destruído de maneira segura de acordo com políticas de destruição da AWS. Para obter mais informações, consulte Dispositivos privados no AWS Device Farm.

Gerenciamento de chaves

Atualmente, o Device Farm não oferece nenhum gerenciamento de chaves externas para criptografia de dados, em repouso ou em trânsito.

Privacidade do tráfego entre redes

O Device Farm pode ser configurado, apenas para dispositivos privados, para usar os endpoints da Amazon VPC para se conectar aos seus recursos na AWS. O acesso a qualquer AWS infraestrutura não pública associada à sua conta (por exemplo, EC2 instâncias da Amazon sem um endereço IP público) deve usar um endpoint Amazon VPC. Independentemente da configuração do endpoint da VPC, o Device Farm isola seu tráfego de outros usuários em toda a rede do Device Farm.

Não é garantido que suas conexões fora da AWS rede sejam seguras ou protegidas, e é sua responsabilidade proteger todas as conexões de internet que seus aplicativos fizerem.

Resiliência em AWS Device Farm

A infraestrutura AWS global é construída em torno de AWS regiões e zonas de disponibilidade. AWS As regiões fornecem várias zonas de disponibilidade fisicamente separadas e isoladas, conectadas a redes de baixa latência, alta taxa de transferência e alta redundância. Com as zonas de disponibilidade, é possível projetar e operar aplicações e bancos de dados que automaticamente executam o failover entre as zonas sem interrupção. As zonas de disponibilidade são altamente disponíveis, tolerantes a falhas e escaláveis que uma ou várias infraestruturas de data center tradicionais.

Para obter mais informações sobre AWS regiões e zonas de disponibilidade, consulte Infraestrutura AWS global.

Como o Device Farm está disponível somente na região `us-west-2`, é altamente recomendável que você implemente processos de backup e recuperação. O Device Farm não deve ser a única fonte de qualquer conteúdo enviado.

O Device Farm não oferece garantias quanto à disponibilidade de dispositivos públicos. Esses dispositivos são inseridos e retirados do grupo de dispositivos públicos dependendo de diversos fatores, como a taxa de falhas e o status de quarentena. Não recomendamos que você dependa da disponibilidade de nenhum dispositivo do grupo de dispositivos públicos.

Segurança da infraestrutura em AWS Device Farm

Como serviço gerenciado, AWS Device Farm é protegido pela segurança de rede AWS global. Para obter informações sobre serviços AWS de segurança e como AWS proteger a infraestrutura, consulte AWS Cloud Security. Para projetar seu AWS ambiente usando as melhores práticas de segurança de infraestrutura, consulte Proteção de infraestrutura no Security Pillar AWS Well-Architected Framework.

Você usa chamadas de API AWS publicadas para acessar o Device Farm pela rede. Os clientes devem oferecer compatibilidade com:

- Transport Layer Security (TLS). Exigimos TLS 1.2 e recomendamos TLS 1.3.
- Conjuntos de criptografia com perfect forward secrecy (PFS) como DHE (Ephemeral Diffie-Hellman) ou ECDHE (Ephemeral Elliptic Curve Diffie-Hellman). A maioria dos sistemas modernos, como Java 7 e versões posteriores, comporta esses modos.

Além disso, as solicitações devem ser assinadas usando um ID da chave de acesso e uma chave de acesso secreta associada a uma entidade principal do IAM. Ou é possível usar o AWS Security Token Service (AWS STS) para gerar credenciais de segurança temporárias para assinar solicitações.

Segurança da infraestrutura para teste de dispositivos físicos

Os dispositivos são fisicamente separados durante o teste de dispositivos físicos. O isolamento da rede impede a comunicação entre dispositivos por meio de redes sem fios.

Os dispositivos públicos são compartilhados, e o Device Farm faz o melhor esforço para manter os dispositivos seguros ao longo do tempo. Determinadas ações, como tentativas para adquirir direitos de administrador completos em um dispositivo (uma prática referida como enraizamento ou fuga), faz com que os dispositivos públicos sejam colocados em quarentena. Eles são removidos do grupo público automaticamente e transferidos para a análise manual.

Os dispositivos privados só podem ser acessados por AWS contas explicitamente autorizadas a fazer isso. O Device Farm isola fisicamente esses dispositivos de outros dispositivos e os mantém em uma rede separada.

Em dispositivos gerenciados de forma privada, os testes podem ser configurados para usar um endpoint Amazon VPC para proteger conexões dentro e fora da sua conta. AWS

Segurança da infraestrutura para teste de navegador de desktop

Quando você usa o recurso de teste de navegador de desktop, todas as sessões de teste são separadas umas das outras. As instâncias do Selenium não podem se comunicar de forma cruzada sem um terceiro intermediário, externo a. AWS

Todo o tráfego para os WebDriver controladores Selenium deve ser feito por meio do endpoint HTTPS gerado com. `createTestGridUrl`

Você é responsável por garantir que cada instância de teste do Device Farm tenha acesso seguro aos recursos que testa. Por padrão, as instâncias de teste do navegador de desktop do Device Farm têm acesso à Internet pública. Quando você anexa sua instância a uma VPC, ela se comporta como qualquer outra EC2 instância, com acesso aos recursos determinados pela configuração da VPC e seus componentes de rede associados. A AWS fornece grupos de segurança e listas de controle de acesso (ACLs) de rede para aumentar a segurança em sua VPC. Os grupos de segurança controlam o tráfego de entrada e saída de seus recursos, e a rede ACLs controla o tráfego de entrada e saída de suas sub-redes. Os grupos de segurança fornecem controle de acesso suficiente para a maioria das sub-redes. Você pode usar a rede ACLs se quiser uma camada adicional de segurança para sua VPC. Para obter diretrizes gerais sobre as melhores práticas de segurança ao usar a Amazon VPCs, consulte as melhores práticas de segurança para sua VPC no Guia do usuário da Amazon Virtual Private Cloud.

Análise e gerenciamento de vulnerabilidades de configuração no Device Farm

O Device Farm permite que você execute software que não é ativamente mantido ou corrigido pelo fornecedor, como o fornecedor do sistema operacional, o fornecedor do hardware ou a operadora de telefonia. A Device Farm se esforça ao máximo para manter o software atualizado, mas não garante que qualquer versão específica do software em um dispositivo físico esteja atualizada, permitindo que softwares potencialmente vulneráveis sejam colocados em uso.

Por exemplo, se um teste for realizado em um dispositivo executando o Android 4.4.2, o Device Farm não garante que o dispositivo tenha sido corrigido contra a vulnerabilidade no Android conhecida como. StageFright É responsabilidade do fornecedor (e às vezes da operadora) do dispositivo fornecer atualizações de segurança para os dispositivos. Não há garantias de que um aplicativo mal-intencionado que use essa vulnerabilidade seja capturado pela nossa quarentena automatizada.

Os dispositivos privados são mantidos de acordo com seu contrato com AWS.

O Device Farm se esforça ao máximo para impedir que as aplicações do cliente realizem ações como root ou jailbreak. O Device Farm remove os dispositivos que estão em quarentena do pool público até que sejam revisados manualmente.

Você é responsável por manter atualizadas todas as bibliotecas ou versões de software que usar em seus testes, como wheels do Python e gemas do Ruby. O Device Farm recomenda que você atualize suas bibliotecas de teste.

Esses recursos podem ajudar a manter as dependências de teste atualizadas:

- Para obter informações sobre como proteger gemas Ruby, consulte Práticas de segurança no RubyGems site.
- Para obter informações sobre o pacote de segurança usado pela Pipenv e aprovado pela Python Packaging Authority para verificar seu gráfico de dependências em busca de vulnerabilidades conhecidas, consulte Detecção de vulnerabilidades de segurança em. GitHub
- Para obter informações sobre o verificador de dependência Maven do Open Web Application Security Project (OWASP), consulte OWASP no site da DependencyCheckOWASP.

É importante lembrar que, mesmo que um sistema automatizado não acredite que haja problemas de segurança conhecidos, isso não significa que não haja problemas de segurança. Sempre tenha

cuidado ao usar bibliotecas ou ferramentas de terceiros e verifique as assinaturas criptográficas quando possível ou razoável.

Resposta a incidentes no Device Farm

O Device Farm monitora continuamente os dispositivos em busca de comportamentos que possam indicar problemas de segurança. Se AWS for informado de um caso em que os dados do cliente, como resultados de testes ou arquivos gravados em um dispositivo público, podem ser acessados por outro cliente, ele AWS entra em contato com os clientes afetados, de acordo com as políticas padrão de alerta e emissão de relatórios de incidentes usadas em todos AWS os serviços.

Registrar em log e monitorar no Device Farm

Esse serviço oferece suporte AWS CloudTrail, que é um serviço que registra AWS chamadas para você Conta da AWS e entrega arquivos de log em um bucket do Amazon S3. Usando as informações coletadas por CloudTrail, você pode determinar quais solicitações foram feitas com sucesso Serviços da AWS, quem fez a solicitação, quando ela foi feita e assim por diante. Para saber mais sobre CloudTrail, inclusive como ativá-lo e encontrar seus arquivos de log, consulte o Guia AWS CloudTrail do usuário.

Para obter informações sobre como usar CloudTrail com o Device Farm, consulteRegistro de chamadas de API do AWS Device Farm com AWS CloudTrail.

Práticas recomendadas de segurança para o Device Farm

O Device Farm oferece vários recursos de segurança que devem ser considerados ao desenvolver e implementar suas próprias políticas de segurança. As práticas recomendadas a seguir são diretrizes gerais e não representam uma solução completa de segurança. Como essas práticas recomendadas podem não ser adequadas ou suficientes para o seu ambiente, trate-as como considerações úteis em vez de prescrições.

- Conceda a qualquer sistema de integração contínua (CI) que você usar o mínimo de privilégios possível no IAM. Considere o uso de credenciais temporárias para cada teste de sistema de CI para que, mesmo que um sistema de CI esteja comprometido, ele não possa fazer solicitações falsas. Para obter mais informações sobre credenciais temporárias, consulte o Guia do usuário do IAM.

- Use comandos adb em um ambiente de teste personalizado para limpar qualquer conteúdo criado pelo aplicativo. Para obter mais informações sobre ambientes de teste personalizados, consulte Ambientes de teste personalizados

Limites do AWS Device Farm

A lista a seguir descreve os limites atuais do AWS Device Farm:

- O tamanho máximo do arquivo de um aplicativo que você pode carregar é de 4 GB.
- Não existe limite quanto ao número de dispositivos que você pode incluir em uma execução de teste. No entanto, o número máximo de dispositivos que o Device Farm testará simultaneamente durante uma execução de teste é cinco. (Esse número poderá ser aumentado mediante solicitação.)
- Não há limite para o número de execuções que você pode programar.
- Há um limite de 150 minutos de duração para uma sessão de acesso remoto.
- Há um limite de 150 minutos para a duração de uma execução de teste automatizado.
- O número máximo de trabalhos em andamento, incluindo trabalhos pendentes em fila em sua conta, é 250. Esse é um limite flexível.
- Não há limite para o número de dispositivos que você pode incluir em um teste. O número de dispositivos (trabalhos) que podem executar seus testes paralelamente a qualquer momento é igual à simultaneidade no nível da sua conta. A simultaneidade padrão em nível de conta para uso medido no Device Farm é cinco.

  O limite de simultaneidade medido pode ser aumentado mediante solicitação até um determinado limite, dependendo do caso de uso. A simultaneidade padrão no nível da conta para uso ilimitado é igual ao número de slots nos quais você está inscrito nessa plataforma.

  Para obter mais informações sobre os limites de simultaneidade medidos padrão ou as cotas em geral, consulte a página Cotas.
- O Device Farm segue um algoritmo de token-bucket para reduzir as taxas de chamadas da API. Por exemplo, imagine criar um balde com fichas. Cada token representa uma transação, e uma chamada de API usa um token. Os tokens são adicionados ao bucket a uma taxa fixa (por exemplo, 10 tokens por segundo), e o bucket tem uma capacidade máxima (por exemplo, 100 tokens). Quando uma solicitação ou pacote chega, ele deve reivindicar um token do bucket para ser processado. Se houver tokens suficientes, a solicitação será autorizada e os tokens serão removidos. Se não houver tokens suficientes, a solicitação será atrasada ou cancelada, dependendo da implementação.

  No Device Farm, é assim que o algoritmo é implementado:

- As solicitações da API Burst são o número máximo de solicitações às quais o serviço pode responder para uma API específica em um ID de conta de cliente especificado. Em outras palavras, essa é a capacidade do balde. Você pode chamar a API quantas vezes houver tokens restantes no bucket, e cada solicitação consome um token.
- A taxa Transactions-per-second (TPS) é a taxa mínima na qual suas solicitações de API podem ser executadas. Em outras palavras, essa é a taxa na qual o bucket é reabastecido com tokens por segundo. Por exemplo, se uma API tiver um número de intermitência de dez, mas um TPS de um, você poderá chamá-la dez vezes instantaneamente. No entanto, o bucket só recuperaria tokens a uma taxa de um token por segundo, resultando na redução para uma chamada por segundo, a menos que você parasse de chamar a API para permitir que o bucket fosse reabastecido.

Aqui estão as tarifas do Device Farm APIs:

- Para List e Get APIs, a capacidade de solicitações da API Burst é `50` e a taxa Transactions-per-second (TPS) é. `10`
- Para todas as outras APIs, a capacidade de solicitações da API Burst é`10`, e a taxa Transactions-per-second (TPS) é. `1`

Ferramentas e plug-ins para o AWS Device Farm

Esta seção contém links e informações sobre como trabalhar com as ferramentas e os plug-ins do AWS Device Farm. Você pode encontrar plug-ins do Device Farm no AWS Labs em GitHub.

Se você é um desenvolvedor Android, também temos um aplicativo de amostra do AWS Device Farm para Android ativado GitHub. Você pode usar a aplicação e os testes de exemplo como referência para seus próprios scripts de teste do Device Farm.

Tópicos

- Integrando o Device Farm com um servidor Jenkins CI
- Integrando o Device Farm com um sistema de compilação Gradle

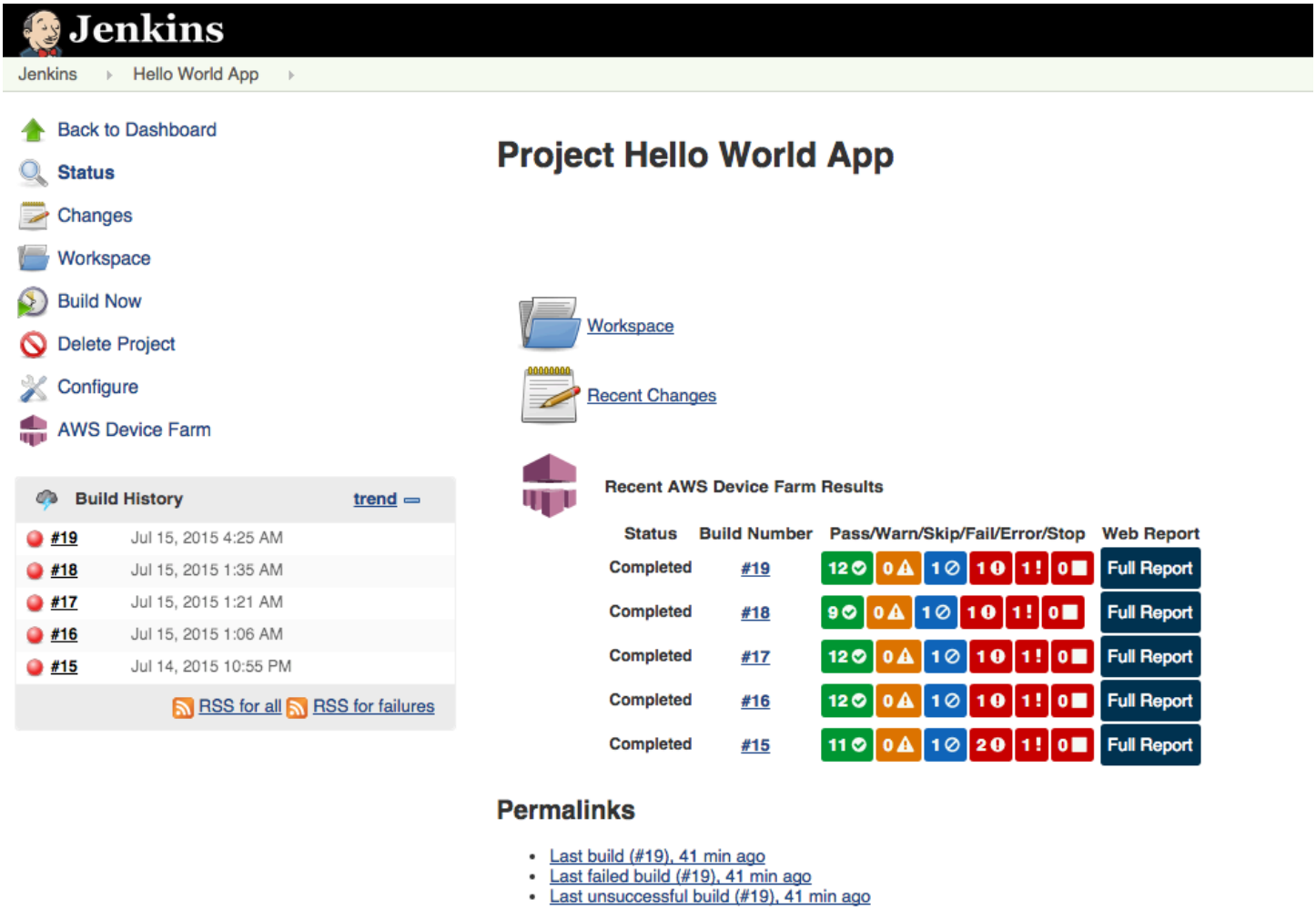
Integrando o Device Farm com um servidor Jenkins CI

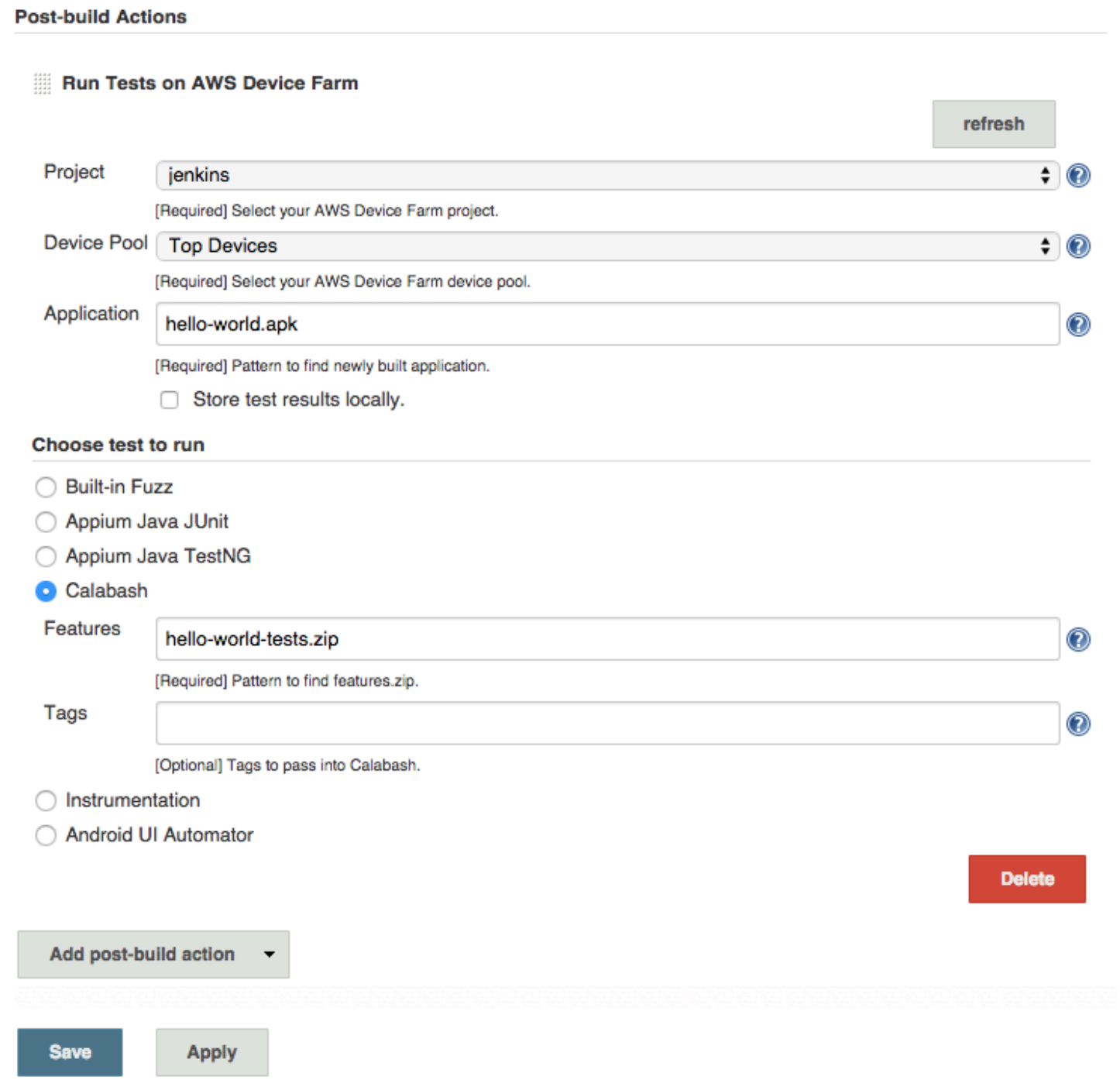
O plug-in Jenkins CI fornece a funcionalidade do AWS Device Farm a partir do seu próprio servidor de integração contínua (CI) Jenkins. Para obter mais informações, consulte Jenkins (software).

> **Note**
>
> Para baixar o plug-in Jenkins, acesse GitHube siga as instruções emEtapa 1: Instalando o plug-in Jenkins CI para o AWS Device Farm.

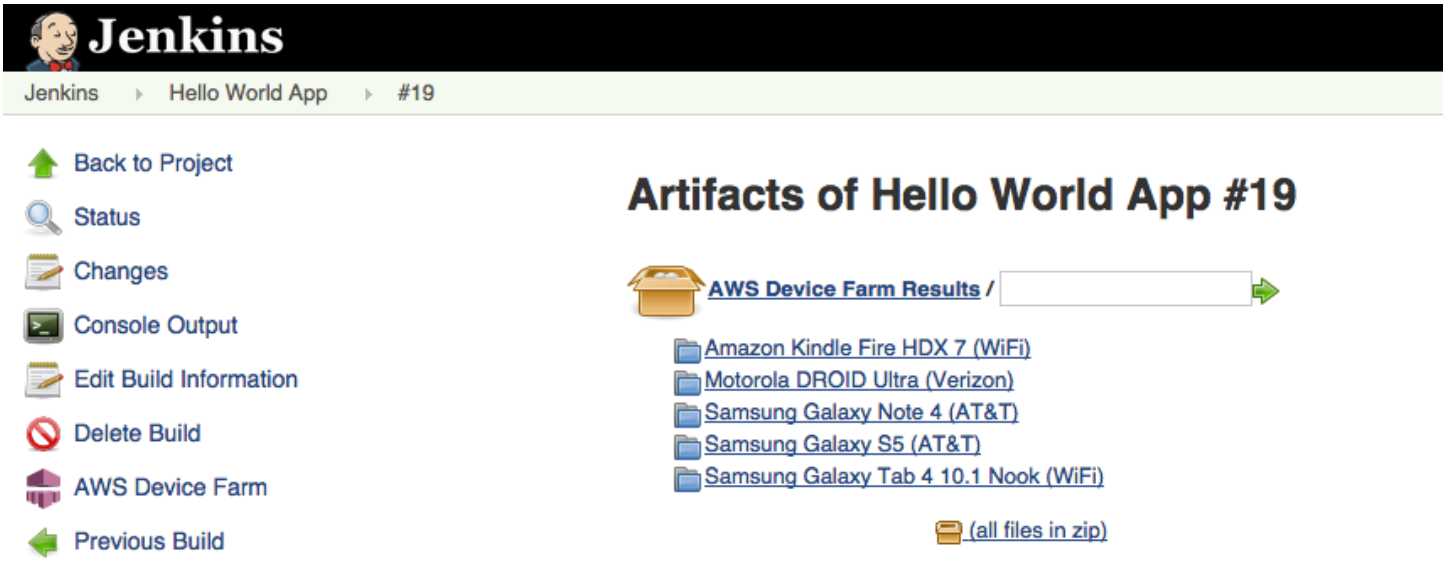
Esta seção contém uma série de procedimentos para configurar e usar o plug-in Jenkins CI com o AWS Device Farm.

As imagens a seguir mostram os recursos do plug-in Jenkins CI.


Jenkins
Jenkins ▸ Hello World App ▸
Back to Dashboard
Status
Changes
Workspace
Build Now
Delete Project
Configure
AWS Device Farm
Build History trend
#19 Jul 15, 2015 4:25 AM
#18 Jul 15, 2015 1:35 AM
#17 Jul 15, 2015 1:21 AM
#16 Jul 15, 2015 1:06 AM
#15 Jul 14, 2015 10:55 PM
RSS for all RSS for failures
Project Hello World App
Workspace
Recent Changes
Recent AWS Device Farm Results
Status Build Number Pass/Warn/Skip/Fail/Error/Stop Web Report
Completed #19 12 0 1 1 1 0 Full Report
Completed #18 9 0 1 1 1 0 Full Report
Completed #17 12 0 1 1 1 0 Full Report
Completed #16 12 0 1 1 1 0 Full Report
Completed #15 11 0 1 2 1 0 Full Report
Permalinks
Last build (#19), 41 min ago
Last failed build (#19), 41 min ago
Last unsuccessful build (#19), 41 min ago



Esse plug-in também pode abrir todos os artefatos de teste (logs, capturas de tela etc.) localmente:



Tópicos

- Dependências
- Etapa 1: Instalando o plug-in Jenkins CI para o AWS Device Farm
- Etapa 2: Criar um AWS Identity and Access Management usuário para seu plug-in Jenkins CI para AWS Device Farm
- Etapa 3: Configurar o plug-in Jenkins CI pela primeira vez no AWS Device Farm
- Etapa 4: usar o plug-in em um trabalho do Jenkins

Dependências

O plug-in Jenkins CI requer o AWS Mobile SDK 1.10.5 ou posterior. Para obter mais informações e instalar o SDK, consulte AWS Mobile SDK.

Etapa 1: Instalando o plug-in Jenkins CI para o AWS Device Farm

Há duas opções para instalar o plug-in de integração contínua (CI) do Jenkins para o AWS Device Farm. Você pode procurar o plug-in na caixa de diálogo Available Plugins (Plug-ins disponíveis) na interface do usuário da web do Jenkins ou você pode fazer download do arquivo `hpi` e instalá-lo por meio do Jenkins.

Instalação por meio da interface de usuário do Jenkins

1. Encontre o plug-in na interface do usuário do Jenkins, escolha Manage Jenkins (Gerenciar Jenkins), Manage Plugins (Gerenciar plug-ins) e Available (Disponível).
2. Pesquise por aws-device-farm.
3. Instale o plug-in AWS Device Farm.
4. Verifique se o plug-in pertence ao usuário `Jenkins`.
5. Reinicie o Jenkins.

Faça download do plug-in.

1. Baixe o `hpi` arquivo diretamente de http://updates.jenkins-ci. org/latest/aws-device-farm.hpi.
2. Verifique se o plug-in pertence ao usuário `Jenkins`.
3. Instale o plug-in usando uma das seguintes opções:
   - Faça upload do plug-in escolhendo Manage Jenkins (Gerenciar Jenkins), Manage Plugins (Gerenciar plug-ins), Advanced (Avançado) e Upload plugin (Fazer upload do plug-in).
   - Coloque o arquivo `hpi` no diretório do plug-in Jenkins (normalmente `/var/lib/jenkins/plugins`).
4. Reinicie o Jenkins.

Etapa 2: Criar um AWS Identity and Access Management usuário para seu plug-in Jenkins CI para AWS Device Farm

Recomendamos que você não use sua conta AWS root para acessar o Device Farm. Em vez disso, crie um novo usuário AWS Identity and Access Management (IAM) (ou use um usuário do IAM existente) em sua AWS conta e, em seguida, acesse o Device Farm com esse usuário do IAM.

Para criar um usuário do IAM, consulte Creating an IAM User (AWS Management Console). Lembre-se de gerar uma chave de acesso para cada usuário e fazer download ou salvar as credenciais de segurança do usuário. Você precisará das credenciais posteriormente.

Dê ao usuário do IAM permissão para acessar o Device Farm

Para dar permissão ao usuário do IAM para acessar o Device Farm, crie uma nova política de acesso no IAM e atribua a política de acesso ao usuário do IAM da seguinte forma.

> **Note**
>
> A conta AWS raiz ou o usuário do IAM que você usa para concluir as etapas a seguir deve ter permissão para criar a seguinte política do IAM e anexá-la ao usuário do IAM. Para obter mais informações, consulte Trabalhar com políticas.

Para criar a política de acesso no IAM

1. Abra o console do IAM em https://console.aws.amazon.com/iam/.
2. Selecione Políticas.
3. Escolha Criar política. (Se aparecer um botão Get Started, selecione-o e, em seguida, Create Policy.)
4. Próximo a Create Your Own Policy, escolha Select.
5. Em Policy Name (Nome da política), digite um nome para política (por exemplo, **`AWSDeviceFarmAccessPolicy`**).
6. Em Descrição, digite uma descrição que ajude a associar esse usuário do IAM ao projeto do Jenkins.
7. Em Policy Document (Documento da política), digite a seguinte declaração:

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

8. Escolha Criar política.

Para atribuir a política de acesso ao usuário do IAM

1. Abra o console do IAM em https://console.aws.amazon.com/iam/.
2. Selecione Usuários.

3. Escolha o usuário do IAM ao qual você atribuirá a política de acesso.
4. Na área Permissions (Permissões), em Managed Policies (Políticas gerenciadas), escolha Attach Policy (Anexar política).
5. Selecione a política que você acabou de criar (por exemplo, AWSDeviceFarmAccessPolicy).
6. Escolha Attach Policy.

Etapa 3: Configurar o plug-in Jenkins CI pela primeira vez no AWS Device Farm

A primeira vez em que você executar o servidor Jenkins, precisará configurar o sistema conforme a seguir.

> **Note**
>
> Se estiver usando slots de dispositivos, o recurso de slots para dispositivo estará desativado por padrão.

1. Faça login na interface de usuário web do Jenkins.
2. No lado esquerdo da tela, escolha Manage Jenkins (Gerenciar Jenkins).
3. Escolha Configure System (Configurar sistema).
4. Role para baixo até o cabeçalho AWS Device Farm.
5. Copie suas credenciais de segurança de Criação de um usuário do IAM para seu plug-in Jenkins CI e cole o ID da chave de acesso e a chave de acesso secreta nas respectivas caixas.
6. Escolha Salvar.

Etapa 4: usar o plug-in em um trabalho do Jenkins

Assim que você tiver instalado o plug-in Jenkins, siga estas instruções para usar o plug-in em um trabalho do Jenkins.

1. Faça login na interface de usuário web do Jenkins.
2. Clique no trabalho que você deseja editar.
3. No lado esquerdo da tela, escolha Configurar.

4. Role para baixo até o cabeçalho Ações pós-compilação.
5. Clique em Adicionar ação pós-compilação e selecione Executar testes no AWS Device Farm.
6. Selecione o projeto que você deseja usar.
7. Selecione o grupo de dispositivos que você deseja usar.
8. Selecione se você gostaria de ter os artefatos de teste (como logs e capturas de tela) arquivados localmente.
9. Em Aplicativo, preencha o caminho do aplicativo compilado.
10. Selecione o teste que deseja executar e preencha todos os campos obrigatórios.
11. Escolha Salvar.

Integrando o Device Farm com um sistema de compilação Gradle

O plug-in Device Farm Gradle fornece integração do AWS Device Farm com o sistema de compilação Gradle no Android Studio. Para obter mais informações, consulte Gradle.

> **Note**
>
> Para baixar o plug-in do Gradle, acesse GitHube siga as instruções emCriação do plug-in Gradle do Device Farm.

O plug-in Gradle do Device Farm fornece a funcionalidade do Device Farm em seu ambiente do Android Studio. Você pode iniciar testes em telefones e tablets Android reais hospedados pelo Device Farm.

Esta seção contém uma série de procedimentos para configurar e usar o plug-in Gradle do Device Farm.

Tópicos

- Dependências
- Etapa 1: Criação do plug-in Gradle do AWS Device Farm
- Etapa 2: Configurar o plug-in Gradle do AWS Device Farm
- Etapa 3: Gerar um usuário do IAM no plug-in Device Farm Gradle
- Etapa 4: Configuração dos tipos de teste

Dependências

Runtime

- O plug-in Device Farm Gradle requer o AWS Mobile SDK 1.10.15 ou posterior. Para obter mais informações e instalar o SDK, consulte AWS Mobile SDK.
- Android tools builder test api 0.5.2
- Apache Commons Lang3 3.3.4

For Unit Tests (Para testes de unidade)

- Testng 6.8.8
- Jmockit 1.19
- Android gradle tools 1.3.0

Etapa 1: Criação do plug-in Gradle do AWS Device Farm

Esse plug-in fornece integração do AWS Device Farm com o sistema de compilação Gradle no Android Studio. Para obter mais informações, consulte Gradle.

> **Note**
>
> A criação desse plug-in é opcional. O plug-in é publicado por meio do Maven Central. Se deseja permitir que o Gradle faça download do plug-in diretamente, ignore esta etapa e vá para Etapa 2: Configurar o plug-in Gradle do AWS Device Farm.

Para criar o plug-in

1. Acesse GitHube clone o repositório.
2. Crie o plug-in usando `gradle install`.

   O plug-in é instalado no seu repositório maven local.

Próxima etapa: Etapa 2: Configurar o plug-in Gradle do AWS Device Farm

Etapa 2: Configurar o plug-in Gradle do AWS Device Farm

Se você ainda não tiver feito isso, clone o repositório e instale o plug-in usando o procedimento descrito em: Criação do plug-in Gradle do Device Farm.

Para configurar o plug-in Gradle do AWS Device Farm

1. Adicione o artefato do plug-in à sua lista de dependências em `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. Configure o plug-in em seu arquivo `build.gradle`. A configuração específica de teste a seguir deve servir de guia:

```
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. Execute o teste do Device Farm usando a seguinte tarefa: `gradle devicefarmUpload`.

   A saída da compilação imprimirá um link para o console do Device Farm, onde você poderá monitorar a execução do teste.

Próxima etapa: Geração de um usuário do IAM no plug-in Device Farm Gradle

Etapa 3: Gerar um usuário do IAM no plug-in Device Farm Gradle

AWS Identity and Access Management (IAM) ajuda você a gerenciar permissões e políticas para trabalhar com AWS recursos. Este tópico orienta você na geração de um usuário do IAM com permissões para acessar os recursos do AWS Device Farm.

Se ainda não tiver feito isso, conclua as etapas 1 e 2 antes de gerar um usuário do IAM.

Recomendamos que você não use sua conta AWS root para acessar o Device Farm. Em vez disso, crie um usuário do IAM (ou use um usuário do IAM existente) em sua conta da AWS e acesse o Device Farm com esse usuário do IAM.

> **Note**
>
> A conta AWS raiz ou o usuário do IAM que você usa para concluir as etapas a seguir deve ter permissão para criar a seguinte política do IAM e anexá-la ao usuário do IAM. Para obter mais informações, consulte Trabalhar com políticas.

Para criar um novo usuário com a política de acesso adequada no IAM

1. Abra o console do IAM em https://console.aws.amazon.com/iam/.

2. Selecione Usuários.
3. Escolha Create New Users (Criar novos usuários).
4. Digite o nome de usuário de sua escolha.

   Por exemplo, **`GradleUser`**.
5. Escolha Criar.
6. Escolha Download Credentials (Fazer download de credenciais) e as salve em um local onde você possa recuperá-las facilmente depois.
7. Escolha Fechar.
8. Escolha o nome de usuário na lista.
9. Em Permissions (Permissões), expanda o cabeçalho Inline Policies (Políticas em linha) clicando na seta para baixo à direita.
10. Escolha Clique aqui onde está escrito: Não há políticas em linha para mostrar. Para criar uma, clique aqui.
11. Na tela Definir permissões, escolha Política personalizada.
12. Escolha Selecionar.
13. Dê um nome à política, como **`AWSDeviceFarmGradlePolicy`**.
14. Cole a política a seguir em Documento da política.

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

15. Escolha Aplicar política.

Próxima etapa: Configuração de tipos de teste.

Para obter mais informações, consulte Creating an IAM User (AWS Management Console) ou Configuração.

Etapa 4: Configuração dos tipos de teste

Por padrão, o plug-in do AWS Device Farm Gradle executa o teste Instrumentação para Android e AWS Device Farm. Se deseja executar seus próprios testes ou especificar outros parâmetros, você pode optar por configurar um tipo de teste. Este tópico fornece informações sobre cada tipo de teste disponível e o que você precisa fazer no Android Studio a fim de configurá-lo para uso. Para obter mais informações sobre os tipos de teste disponíveis no Device Farm, consulte Estruturas de teste e testes integrados no AWS Device Farm.

Se ainda não tiver feito isso, execute as etapas de 1 a 3 antes de configurar os tipos de teste.

> **Note**
>
> Se estiver usando slots de dispositivos, o recurso de slots para dispositivo estará desativado por padrão.

Appium

O Device Farm fornece suporte para Appium Java e JUnit TestNG para Android.

- Appium (em Java ()) JUnit
- Appium (em Java [TestNG])

Você pode escolher `useTestNG()` ou `useJUnit()`. `JUnit` é o padrão e não precisa ser especificado explicitamente.

```
appium {
    tests file("path to zip file") // required
    useTestNG() // or useJUnit()
}
```

Integrado: Fuzz

O Device Farm fornece um tipo de teste de fuzz incorporado, que envia aleatoriamente eventos da interface do usuário para dispositivos e, em seguida, relata os resultados.

```
fuzz {
```

```
    eventThrottle 50 // optional default
    eventCount 6000  // optional default
    randomizerSeed 1234 // optional default blank

  }
```

Para obter mais informações, consulte Executando o teste de fuzz integrado do Device Farm (Android e iOS).

Instrumentação

O Device Farm fornece suporte para instrumentação (EspressoJUnit, Robotium ou qualquer teste baseado em instrumentação) para Android. Para obter mais informações, consulte Instrumentação para Android e AWS Device Farm.

Ao executar um teste de instrumentação no Gradle, o Device Farm usa o arquivo `.apk` gerado pelo diretório androidTest como a origem dos testes.

```
instrumentation {

    filter "test filter per developer docs" // optional

}
```

Histórico de documentos do AWS Device Farm

A tabela a seguir descreve as mudanças importantes na documentação desde a última versão deste guia.

Alteração	Descrição	Alterado em
AL2 apoio	O Device Farm agora é compatível com o ambiente de AL2 teste para Android. Saiba mais sobre AL2.	6 de novembro de 2023
Migração de ambientes de teste padrão para personalizados	Guia de migração atualizado para documentar a descontinuação dos testes do modo padrão em dezembro de 2023.	3 de setembro de 2023
Suporte a ENI de VPC	O Device Farm agora permite que dispositivos privados usem o recurso de conectividade ENI de VPC para ajudar os clientes a se conectarem com segurança a seus endpoints privados hospedados na AWS, no software on-premise ou em outro provedor de nuvem. Saiba mais sobre ENI de VPC.	15 de maio de 2023
Atualizações da interface do usuário do Polaris	O console do Device Farm agora é compatível com a estrutura Polaris.	28 de julho de 2021
Suporte ao Python 3	O Device Farm agora é compatível com Python 3 em testes de modo personalizado. Saiba mais sobre como usar o Python 3 nos pacotes de teste: • Appium (Python) • Appium (Python)	20 de abril de 2020
Novas informações de segurança e informações sobre	Para tornar AWS os serviços de proteção mais fáceis e abrangentes, uma nova seção sobre segurança foi criada. Para ler mais, consulte Segurança em AWS Device Farm	27 de março de 2020

Alteração	Descrição	Alterado em
AWS recursos de marcação.	Foi adicionada uma nova seção sobre marcação no Device Farm. Para saber mais sobre marcação, consulte Marcação no Device Farm.	
Remoção do acesso direto ao dispositivo.	O acesso direto a dispositivos (depuração remota em dispositivos privados) não está mais disponível para uso geral. Para consultas sobre a disponibilidade futura do acesso direto de dispositivos, entre em contato conosco.	9 de setembro de 2019
Atualizar a configuração do plug-in Gradle	Uma configuração revisada do plug-in Gradle agora inclui uma versão personalizável da configuração gradle, com parâmetros opcionais comentados. Saiba mais sobre Configuração do plug-in Gradle do Device Farm.	16 de agosto de 2019
Novo requisito para execuções de teste com XCTest	Para execuções de teste que usam a XCTest estrutura, o Device Farm agora exige um pacote de aplicativos criado para testes. Saiba mais sobre the section called “XCTest”.	4 de fevereiro de 2019
Compatibilidade com os tipos de teste do Appium Node.js e do Appium Ruby em ambientes personalizados	Agora você pode executar seus testes nos ambientes de teste personalizados do Appium Node.js e do Appium Ruby. Saiba mais sobre Estruturas de teste e testes integrados no AWS Device Farm.	10 de janeiro de 2019

Alteração	Descrição	Alterado em
Suporte para servidor Appium versão 1.7.2 em ambientes padrão e personalizados. Suporte para versão 1.8.1 usando um arquivo YAML da especificação de teste personalizado em um ambiente de teste personalizado.	Você já pode executar os testes padrão e personalizados em ambos os ambientes de teste com versões do servidor Appium 1,72, 1.71 e 1.6.5. Você também pode executar os testes com versões 1.8.1 e 1.8.0 usando um arquivo YAML da especificação de teste personalizado em um ambiente de teste personalizado. Saiba mais sobre Estruturas de teste e testes integrados no AWS Device Farm.	2 de outubro de 2018
Ambientes de teste personalizados	Com um ambiente de teste personalizado, você pode garantir que seus testes sejam executados da mesma forma que em seu ambiente local. O Device Farm agora fornece suporte para log ao vivo e streaming de vídeo, para que você possa obter feedback instantâneo sobre seus testes que são executados em um ambiente de teste personalizado. Saiba mais sobre Ambientes de teste personalizados no AWS Device Farm.	16 de agosto de 2018
Support para usar o Device Farm como provedor AWS CodePipeline de testes	Agora você pode configurar um pipeline AWS CodePipeline para usar as execuções do AWS Device Farm como ações de teste em seu processo de lançamento. CodePipeline permite que você vincule rapidamente seu repositório às etapas de criação e teste para obter um sistema de integração contínua personalizado de acordo com suas necessidades. Saiba mais sobre Integração do AWS Device Farm em um estágio de CodePipeline teste.	19 de julho de 2018

Alteração	Descrição	Alterado em
Suporte para dispositivos privados	Agora você pode usar dispositivos privados para programar execuções de teste e iniciar sessões de acesso remoto. Você pode gerenciar perfis e configurações para esses dispositivos, criar endpoints da Amazon VPC para testar aplicações privadas e criar sessões de depuração remota. Saiba mais sobre Dispositivos privados no AWS Device Farm.	2 de maio de 2018
Compatibilidade com o Appium 1.6.3	Agora você pode definir uma versão do Appium para testes personalizados do Appium.	21 de março de 2017
Definição do tempo limite de execução de testes	Você pode definir o tempo limite para a execução de um teste ou de todos os testes em um projeto. Saiba mais sobre Definindo o tempo limite de execução para execuções de teste no AWS Device Farm.	9 de fevereiro de 2017
Modelagem de rede	Agora você pode simular conexões e condições de rede para a execução de um teste. Saiba mais sobre Simulação de conexões e condições de rede para suas execuções do AWS Device Farm.	8 de dezembro de 2016
Nova seção de solução de problemas	Agora é possível solucionar problemas de uploads de pacotes de teste usando um conjunto de procedimentos criados para resolver mensagens de erro que podem ser encontradas no console do Device Farm. Saiba mais sobre Solução de problemas de Device Farm.	10 de agosto de 2016
Sessões de acesso remoto	Agora você pode acessar e interagir remotamente com um único dispositivo no console. Saiba mais sobre Acesso remoto.	19 de abril de 2016
Autoatendimento para slots de dispositivo	Agora você pode comprar slots de dispositivos usando a AWS Management Console AWS Command Line Interface , a ou a API. Saiba mais sobre como Comprar um slot de dispositivo no Device Farm.	22 de março de 2016

Alteração	Descrição	Alterado em
Como interromper execuções de teste	Agora você pode interromper as execuções de teste usando a AWS Management Console AWS Command Line Interface, a ou a API. Saiba mais sobre como Interromp endo uma execução no AWS Device Farm.	22 de março de 2016
Novos tipos de teste de XCTest interface do usuário	Agora você pode executar testes personalizados de XCTest interface de usuário em aplicativos iOS. Saiba mais sobre o tipo de teste Integrando a XCTest interface do usuário para iOS com o Device Farm.	8 de março de 2016
Novos tipos de teste do Appium Python	Você já pode executar testes personalizados do Appium Python no Android, no iOS e em aplicativos web. Saiba mais sobre Estruturas de teste e testes integrados no AWS Device Farm.	19 de janeiro de 2016
Tipos de teste de aplicativos web	Agora você pode executar testes personalizados Appium Java e JUnit TestNG em aplicativos web. Saiba mais sobre Testes de aplicativos web no AWS Device Farm.	19 de novembro de 2015
Plug-in Gradle do AWS Device Farm	Saiba mais sobre como instalar e usar o Plug-in Gradle do Device Farm.	28 de setembro de 2015
Novo teste integrado para Android: Explorer	O teste integrado Explorer percorre o aplicativo analisando cada tela como se fosse um usuário final e faz capturas de tela à medida que o examina.	16 de setembro de 2015
Nova opção compatibilidade com iOS	Saiba mais sobre como testar dispositivos iOS e executar testes iOS (inclusive XCTest) emEstruturas de teste e testes integrados no AWS Device Farm.	4 de agosto de 2015
Lançamento público inicial	Essa é a versão pública inicial do Guia do desenvolvedor do AWS Device Farm.	13 de julho de 2015

AWS Glossário

Para obter a AWS terminologia mais recente, consulte o AWS glossário na Glossário da AWS Referência.

As traduções são geradas por tradução automática. Em caso de conflito entre o conteúdo da tradução e da versão original em inglês, a versão em inglês prevalecerá.