Guia do Desenvolvedor

AWS Device Farm

Versão da API 2015-06-23

AWS Device Farm: Guia do Desenvolvedor

Table of Contents

O que é o AWS Device Farm?

O Device Farm é um serviço de teste de aplicativos que você pode usar para testar e interagir com seus aplicativos Android, iOS e Web em telefones e tablets reais e físicos hospedados pelo 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 aplicativos

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 contendo resultados de alto nível, logs de baixo nível, capturas de tela pixel a pixel e dados de desempenho é atualizado.

O Device Farm oferece suporte a 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 Trabalho com tipos de teste 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 Preços 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.

executar

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.

session

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.

Configuração

Para usar o Device Farm, consulteConfiguração.

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

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. Acesse https://portal.aws.amazon.com/billing/signup.
2. Siga as instruções on-line.

   Durante a criação da conta, você vai receber uma chamada telefônica e inserir um código de verificação no 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 atributos na conta. Como prática recomendada de segurança, atribua acesso administrativo a um usuário administrativo e utilize somente o usuário raiz para executar as tarefas que exigem acesso do 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 Criação de um usuário IAM (AWS Management Console).

Etapa 3: Dê 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 segue.

> **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 de IAM, consulte Criação de políticas de IAM no Guia do usuário do IAM.

2. Anexe a política de IAM que você criou ao seu novo usuário. Para obter mais informações sobre como anexar políticas de IAM aos usuários, consulte Adição e remoção de políticas de IAM no Guia do usuário do IAM.

A anexação da 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 você está 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) AWS.

Tópicos

Pré-requisitos

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

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

- Para Android, você precisa de um arquivo .apk (pacote de aplicativos Android). 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 Trabalho com tipos de teste 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 1: criar um projeto

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

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

   O console abre a página de 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 página Automated tests (Testes automatizados), escolha Create a new run (Criar uma nova execução).
2. Na página Escolher aplicativo, em Aplicativo móvel, escolha Escolher arquivo e escolha um arquivo Android (.apk) ou iOS (.ipa) em seu computador. Ou arraste o arquivo do seu computador e solte-o no console.
3. Digite um Nome de execução, como **`my first test`** . Por padrão, o console Device Farm usa o nome do arquivo.

4. Escolha Next (Próximo).
5. Na página Configurar, em Configurar estrutura de teste, escolha uma das estruturas de teste ou suítes de teste integradas. Para ter mais informações sobre cada opção, consulte Tipos e estruturas de teste.
   - Se você ainda não tiver empacotado seus testes para o Device Farm, escolha Built-in: Fuzz para executar uma suíte de testes padrão integrada. Você pode manter os valores padrão para contagem de eventos, aceleração de eventos e semente do Randomizer. 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. Escolha Next (Próximo).
7. Na página Selecionar dispositivos, em Pool de dispositivos, escolha Principais dispositivos.
8. Escolha Next (Próximo).
9. Na página Specify device state (Especificar estado do dispositivo), execute uma das seguintes ações:
   - Para fornecer dados adicionais para o Device Farm usar durante a execução, em Adicionar dados extras, faça upload de um arquivo.zip.
   - Para instalar outros aplicativos para a execução, em Instalar outros aplicativos, faça o upload dos arquivos.apk ou.ipa dos aplicativos. Para alterar a ordem de instalação, arraste e solte os arquivos.
   - Para ativar rádios Wi-Fi, Bluetooth, GPS ou NFC durante a execução, em Definir estados de rádio, marque as caixas de seleção correspondentes.

     > **Note**
     >
     > A configuração do estado do rádio do dispositivo está disponível apenas para testes nativos do Android no momento.

   - Para testar o comportamento específico do local durante a execução, em Localização do dispositivo, especifique as coordenadas predefinidas de latitude e longitude.
   - Para predefinir o idioma e a região do dispositivo para a execução, em Local do dispositivo, escolha um local.
   - Para predefinir o perfil de rede para a execução, em Perfil de rede, escolha um perfil selecionado. Ou escolha Criar perfil de rede para criar seu próprio perfil.

10. Escolha Next (Próximo).
11. Na página Review and start run (Examinar e iniciar execução), escolha Confirm and start run (Confirmar e iniciar 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 Trabalhar com relatórios de teste no Device Farm.

Próximas etapas

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

Compre um slot de dispositivo no Device Farm

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

Tópicos

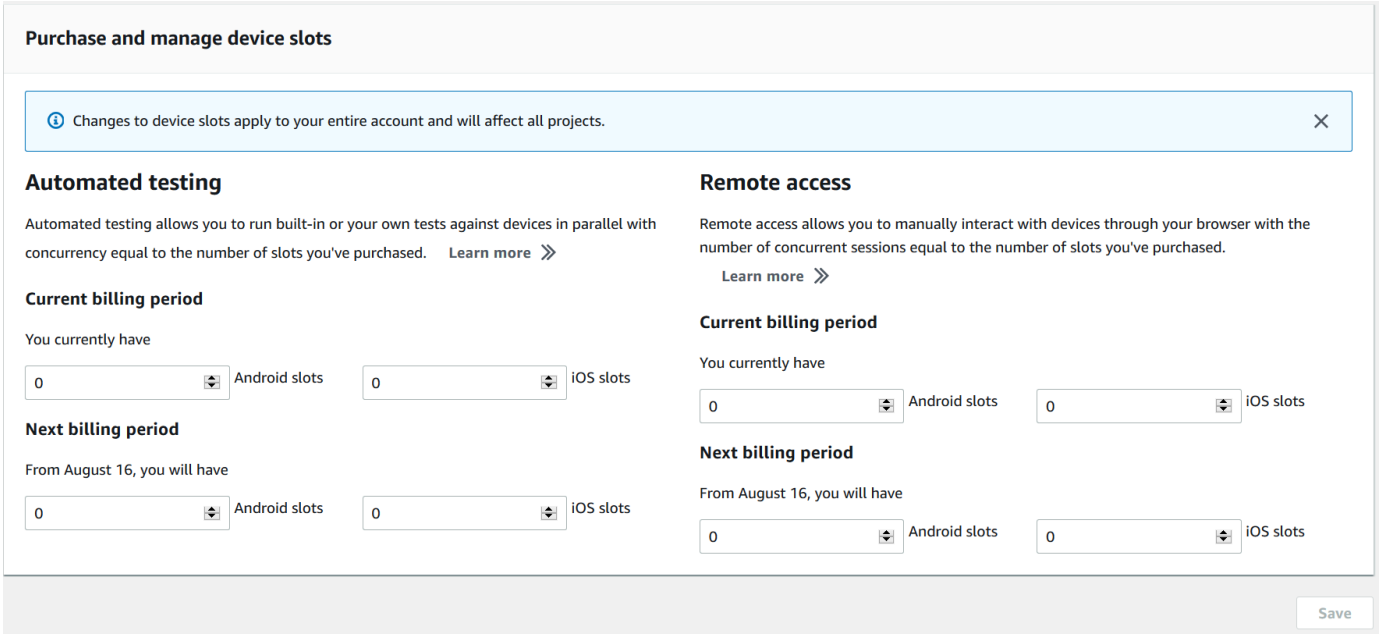
Comprar slots de dispositivos (console)

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. No painel de navegação, escolha Teste de dispositivo móvel e, em seguida, escolha Slots de dispositivo.
3. Na página Comprar e gerenciar 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.

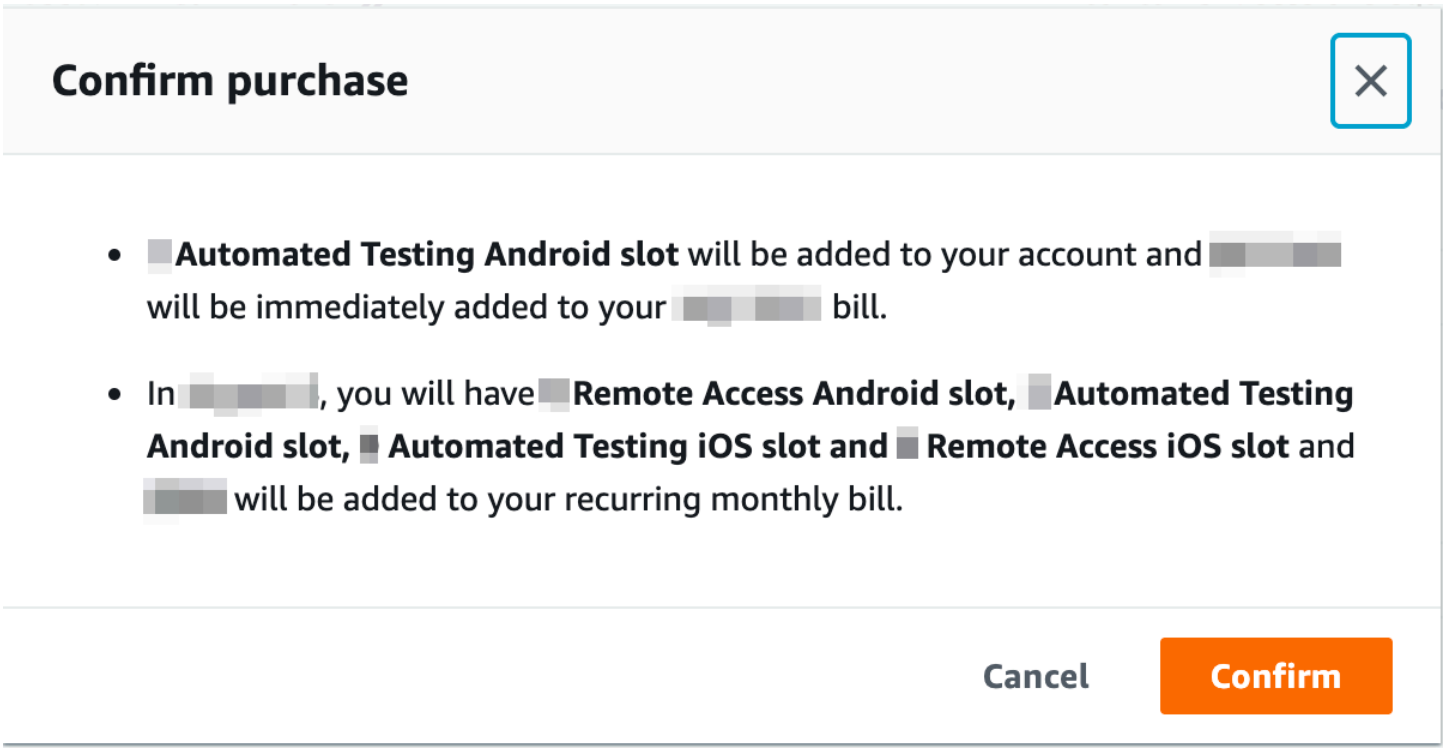
   Conforme você altera o valor do slot, o texto é atualizado dinamicamente com o valor do faturamento. Para obter mais informações, consulte AWS Device Farmpricing.

   > **Important**
   >
   > Se você alterar o número de slots de dispositivo, mas vir uma mensagem de contato ou entre em contato conosco para comprar, sua conta AWS 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 é exibida. Revise as informações e escolha Confirmar para concluir a transação.



Na página Comprar e gerenciar 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 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 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 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 Referência do AWS CLI.

Comprar um slot de dispositivo (API)

1. Chame a operação GetAccountSettings para listar as configurações de conta.
2. Chame a operação ListOfferings para listar as ofertas de slot de dispositivo disponíveis para você.
3. Chame a operação ListOfferingPromotions para listar as promoções de ofertas 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. Chame a operação PurchaseOffering para adquirir uma oferta.
5. Chame a operação GetOfferingStatus 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 .

Conceitos do AWS Device Farm

Esta seção descreve conceitos importantes do Device Farm.

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

Para obter mais informações sobre os tipos de teste compatíveis no Device Farm, consulte Trabalho com tipos de teste 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
- Aplicativos 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 suportadas 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 Trabalhando com dispositivos privados no AWS Device Farm.

Marcas de dispositivo

O Device Farm executa testes em dispositivos móveis e tablets físicos 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.

Aplicativos de dispositivos pré-instalados

Os dispositivos no Device Farm incluem um pequeno número de aplicativos que já estão instalados 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 registros 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 ter mais informações, consulte Trabalhar com ambientes de teste personalizados.

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

Para obter mais informações, consulte Carregamento de uma especificação de teste personalizada usando a AWS CLI e Criar 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 do seu aplicativo, com um conjunto específico de testes, a ser executada em um conjunto específico de dispositivos. A execução produz um relatório que contém informações sobre os resultados da execução. A execução contém um ou mais trabalhos.

Tópicos

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 seus aplicativos 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, registros 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 ter mais informações, consulte Definir o tempo limite de execução para execuções de teste no AWS Device Farm.

Instrumentação de aplicativos

Não é necessário instrumentar seus aplicativos ou fornecer ao Device Farm o código-fonte dos aplicativos. Os aplicativos Android podem ser enviados não modificados. Os aplicativos iOS devem ser compilados com o destino Dispositivo iOS, em vez do simulador.

Nova assinatura de aplicativos nas execuções

Para aplicativos iOS, não é necessário adicionar nenhum UUID do Device Farm ao seu perfil de provisionamento. O Device Farm substitui o perfil de provisionamento incorporado por um perfil curinga e, em seguida, assina novamente o aplicativo. 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 do aplicativo. 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 aplicativos Android, o Device Farm assina novamente o aplicativo. Isso pode interromper qualquer funcionalidade que dependa da assinatura do aplicativo, como a API Android do Google Maps, ou pode acionar a detecção antipirataria ou antiadulteração em produtos como. DexGuard

Aplicativos obscurecidos 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.

Anúncios nas execuções

Recomendamos que você remova os anúncios de seus aplicativos antes de carregá-los 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 obter mais informações, consulte Trabalhar com execuções de teste no AWS Device Farm e Criar uma execução de teste no Device Farm.

Relatórios no AWS Device Farm

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

Tópicos

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, registros 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 ter mais informações, consulte Trabalhar com relatórios de teste no Device Farm.

Sessões no AWS Device Farm

Você pode usar o Device Farm para realizar testes interativos de aplicativos 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 que comportam acesso remoto

O Device Farm oferece suporte a 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 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 seus aplicativos 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 logs de sessão e vídeos capturados por 400 dias e depois os exclui do sistema.

Instrumentação de aplicativos

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

Nova assinatura de aplicativos nas sessões

O Device Farm assina novamente os aplicativos 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 nova assinatura do aplicativo também pode disparar a detecção antipirataria ou adulteração em produtos, como o DexGuard para dispositivos Android.

Aplicativos obscurecidos nas sessões

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

Trabalhando com 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

Criar 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 Mobile Device Testing e, em seguida, escolha Projects.
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 Definir 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 resposta da AWS CLI inclui o nome de recurso da Amazon (ARN) do projeto.

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

  Para obter mais informações, consulte create-project e Referência do AWS CLI.

Criar um projeto (API)

- Chame a API `CreateProject`.

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

Veja a lista de projetos no AWS Device Farm

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

Tópicos

Pré-requisitos

- Crie pelo menos um projeto no Device Farm. Siga as instruções em Criar 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 Mobile Device Testing e escolha Projects.
   - Para projetos de teste de navegadores de desktop, no menu de navegação do Device Farm, escolha Desktop Browser Testing e, em seguida, escolha Projects.

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, consulte Referência do AWS CLI.

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.

Trabalhar com execuções de teste no AWS Device Farm

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

Tópicos

Criar 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.

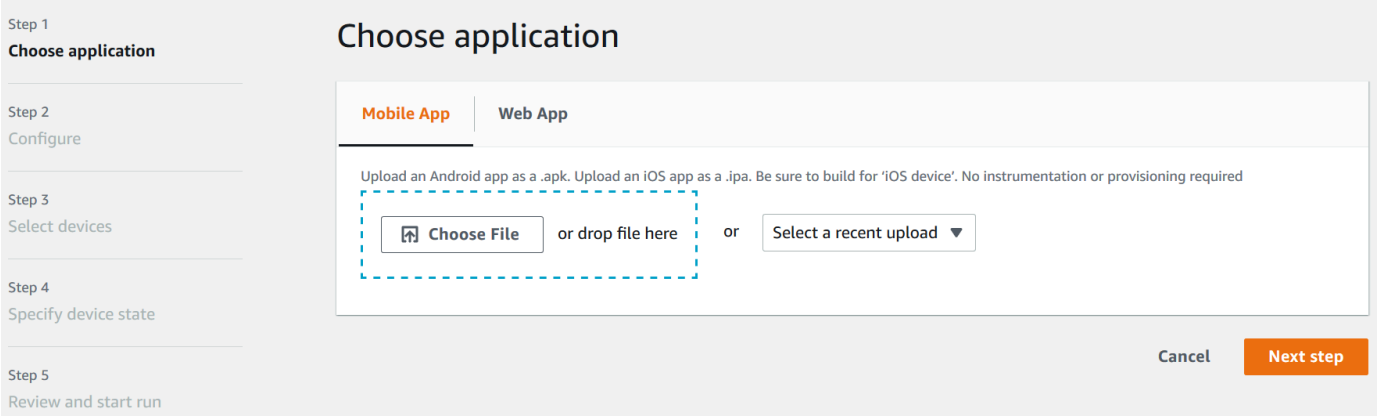
Tópicos

Pré-requisitos

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

Crie 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 Mobile Device Testing e, em seguida, selecione Projetos.
3. Se já tiver um projeto, você poderá fazer upload dos testes para ele. Caso contrário, selecione Novo projeto, insira um Nome de projeto e, em seguida, selecione Criar.
4. Abra o projeto e escolha Criar uma nova execução.
5. Na página Escolher aplicativo, escolha Aplicativo móvel ou Aplicativo da Web.



6. Faça upload do arquivo do aplicativo. Você também pode arrastar e soltar o arquivo ou escolher um upload recente. Se você estiver fazendo upload de um aplicativo iOS, certifique-se de escolher iOS device (Dispositivo iOS), e não um simulador.
7. (Opcional) Em Run name (Nome da execução), insira um nome. Por padrão, o Device Farm usa o nome do arquivo do aplicativo.
8. Escolha Próximo.
9. Na página Configurar, escolha um dos pacotes de testes disponíveis.

> **Note**
>
> Se não tiver testes disponíveis, escolha Integrado: Fuzz para executar um pacote integrado padrão de testes. Se escolher Integrado: Fuzz 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 Trabalho com tipos de teste no AWS Device Farm.

10. Se você não tiver escolhido Integrado: Fuzz, selecione Escolher arquivo e, em seguida, navegue até o arquivo que contém os testes e escolha-o.
11. Para seu ambiente de teste, escolha Executar o teste em nosso ambiente padrão ou Executar o teste em um ambiente personalizado. Para ter mais informações, consulte Ambientes de teste.
12. Se você estiver usando o ambiente de teste padrão, vá para a etapa 13. Se você estiver usando um ambiente de teste personalizado com o arquivo YAML de especificação de teste padrão, vá para a etapa 13.

    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. Se você quiser configurar as opções de captura de dados de desempenho ou gravação de vídeo, escolha Configuração avançada.

    a. Selecione Ativar gravação de vídeo para gravar vídeos durante o teste.

    b. Selecione Ativar captura de dados de desempenho do aplicativo para capturar dados de desempenho do dispositivo.

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

14. Escolha Próximo.
15. Na página Selecionar dispositivos, execute uma das seguintes ações:

    - 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 Criar um grupo 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.

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

16. Escolha Próximo.
17. Na página Specify device state (Especificar estado do dispositivo):
    - 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 um aplicativo adicional para o Device Farm usar durante a execução, ao lado de Instalar outros aplicativos, selecione Escolher arquivo e, em seguida, procure e escolha o arquivo .apk ou .ipa que contém o aplicativo. 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.
18. Escolha Próximo.
19. Na página Revisar e iniciar a execução, você pode especificar o tempo limite de execução do seu teste. Se você estiver usando um número ilimitado de slots de testes, confirme se Executar em slots não medidos está selecionado.
20. Insira um valor ou use a barra deslizante para alterar o tempo limite de execução. Para ter mais informações, consulte Definir o tempo limite de execução para execuções de teste no AWS Device Farm.
21. Escolha Confirmar e iniciar 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 Interromper 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

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 Criar 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: Escolha um pool 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 pools de dispositivos incorporados do Device Farm, como Top Devices, e quaisquer pools 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 ter mais informações, consulte Criar um grupo de dispositivos (AWS CLI).

Etapa 3: Faça upload do seu arquivo de candidatura

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

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

Para ter 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 do aplicativo 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 Faça o upload de objetos usando URLs predefinidas 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: Carregue seu 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 predefinido 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 predefinido 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) Faça 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 suportado. 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 ter mais informações, consulte Ambientes de teste.

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 seu arquivo de especificações de teste usando o URL predefinido do Amazon S3. Este exemplo é usado curl para fazer upload de 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 ter 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 ter 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 ter mais informações, consulte get-run. Para obter informações sobre como usar o Device Farm com o AWS CLI, consulteReferência do AWS CLI.

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 Criar um grupo 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 ter mais informações, consulte Etapa 5: (Opcional) Faça 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 ter mais informações, consulte Relatórios no AWS Device Farm.

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

Definir 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 Device Farm Pricing.

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 Criar 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 Mobile Device Testing e, em seguida, escolha Projects.
3. Se você já tiver um projeto, selecione-o na lista. Caso contrário, escolha Novo projeto, digite 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 Mobile Device Testing e, em seguida, escolha Projects.
3. Se você já tiver um projeto, selecione-o na lista. Caso contrário, escolha Novo projeto, digite 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 Revisar e iniciar execução, para Definir tempo limite de execução, insira um valor ou use a barra deslizante.
7. Escolha Confirmar e iniciar execução.

Simule 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 seus aplicativos 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

- Configure a modelagem de rede ao programar uma execução de teste

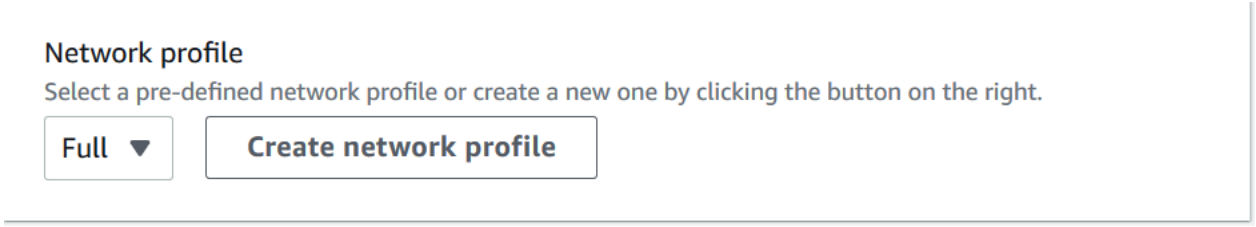
- Criar um perfil de rede
- Altere as condições da rede durante o teste

Configure 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 Criar um projeto no AWS Device Farm.
2. Escolha seu aplicativo 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 Localização e configurações de rede, escolha um perfil de rede ou selecione Criar perfil de rede para criar o seu próprio perfil.

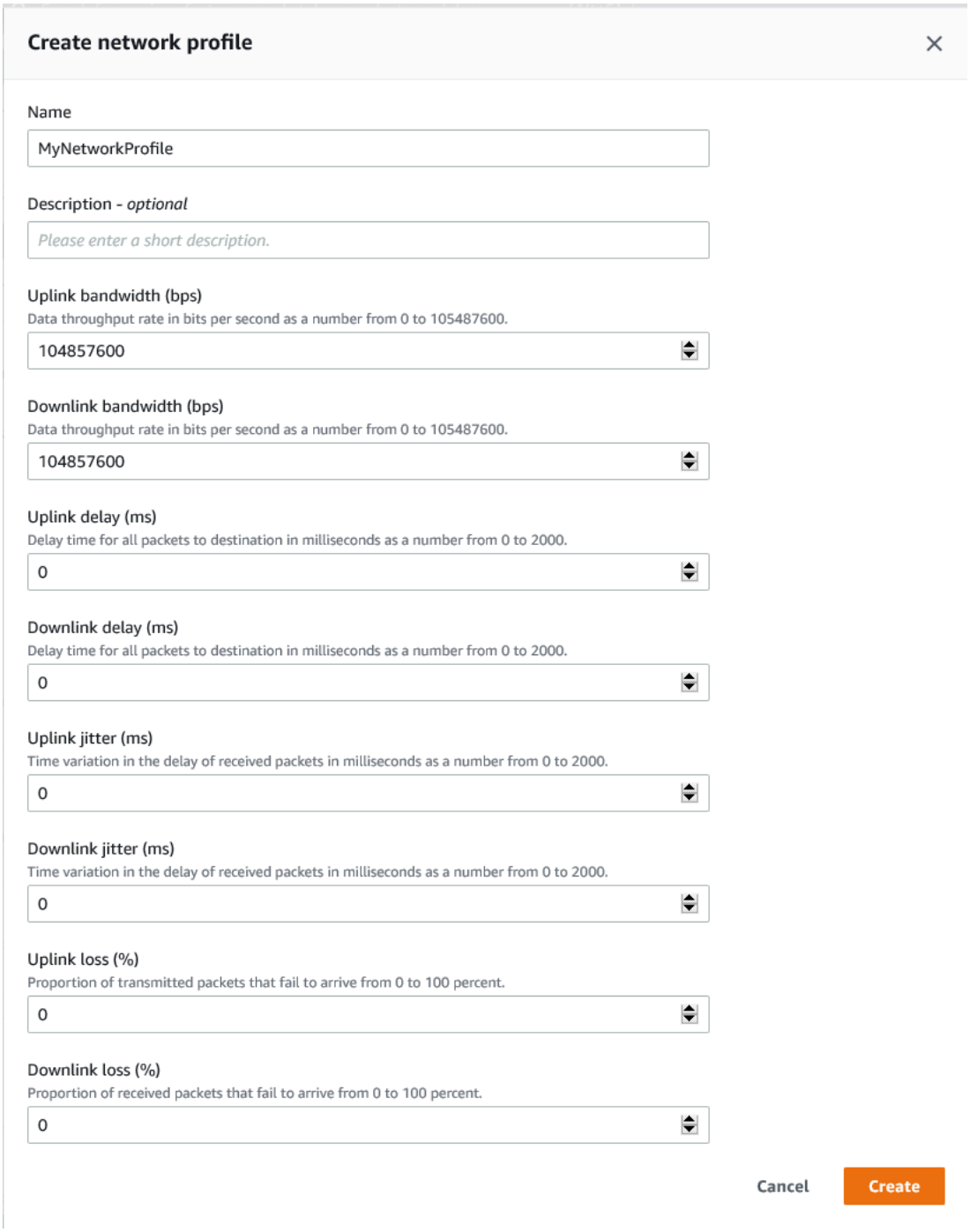


6. Escolha Próximo.
7. Analise e inicie a execução de teste.

Criar um perfil de rede

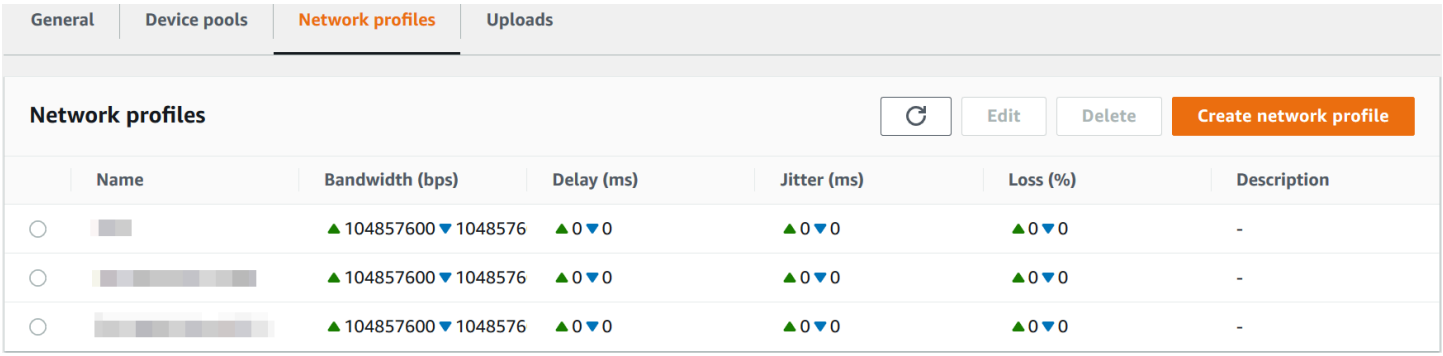
Ao criar uma execução de teste, você pode criar um perfil de rede.

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 que você tiver criado um perfil de rede, você poderá ver e gerenciá-lo na página Configurações do projeto.



Altere 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.

Interromper 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

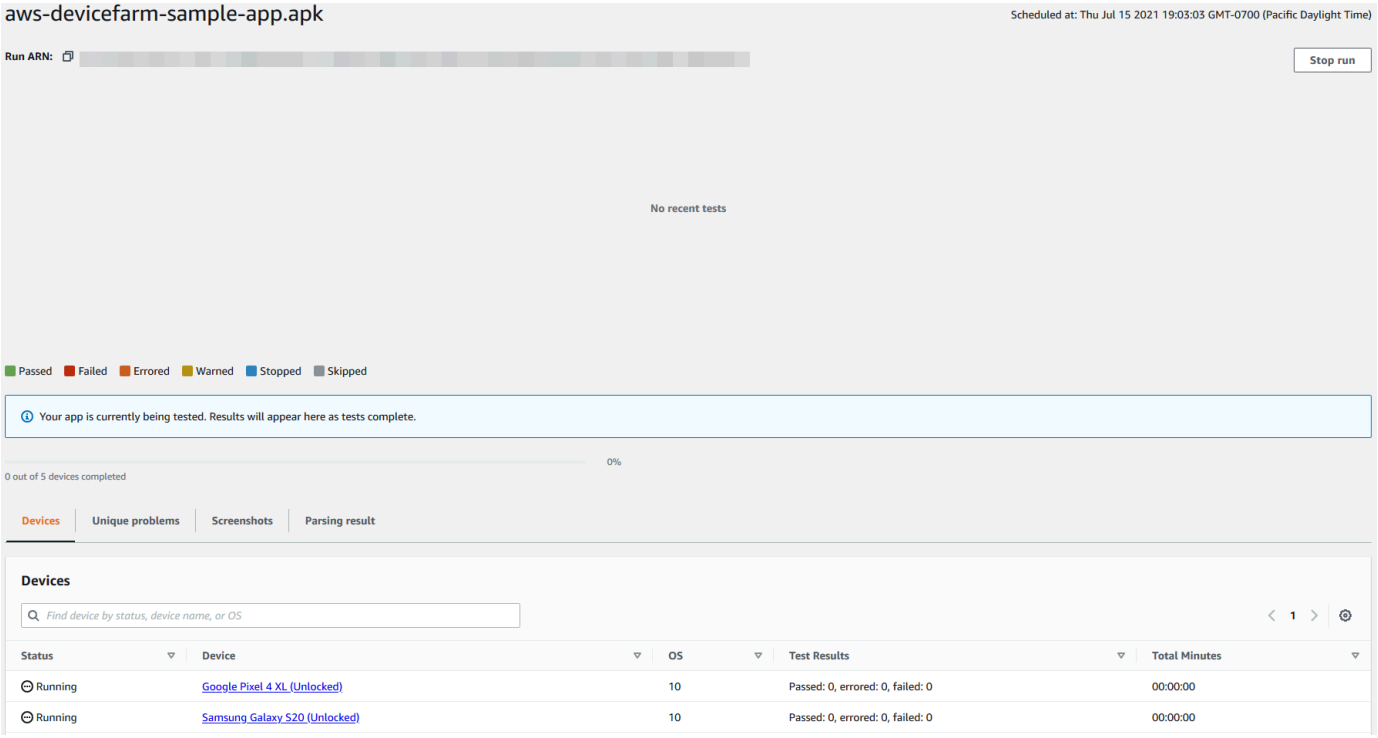
Tópicos

- Parar uma execução (Console)
- Interromper uma execução (AWS CLI)
- Parar uma execução (API)

Parar 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 Mobile Device Testing 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.

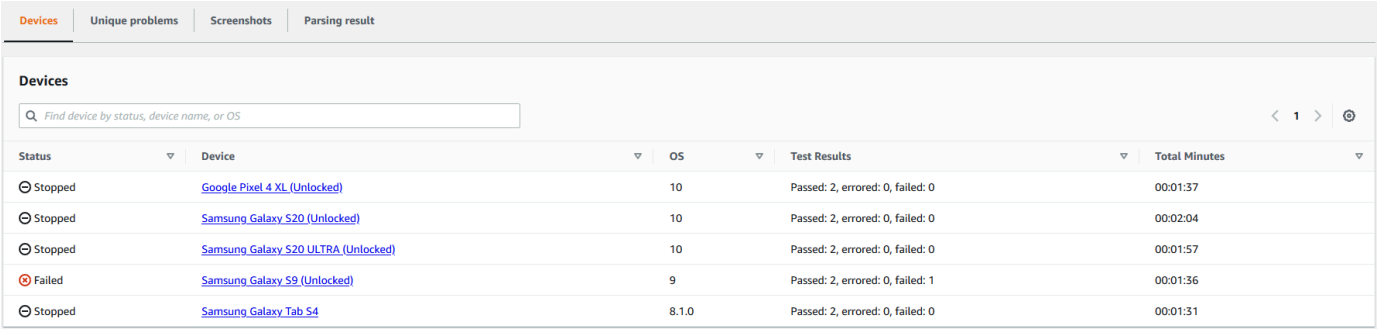


5. Escolha Parar execução.

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 de teste especificada, onde *myARN* é o nome de recurso da Amazon (ARN) da execução de teste.

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

Você deve ver 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 à 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, consulteReferência do AWS CLI.

Parar 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 .

Exibir 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 Mobile Device Testing 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, consulteReferência do AWS CLI.

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.

Criar um grupo 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

- Crie uma execução no console do Device Farm. Siga as instruções em Criar 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 Selecionar dispositivos, escolha Criar pool de dispositivos.
2. Em Nome, digite 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 grupo de dispositivos dinâmicos.

b. Escolha Adicionar 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 por seu valor de tipo, escolha Fator forma.

d. Para Operador (segunda lista suspensa), escolha IGUAL A para incluir dispositivos em que o valor do Campo é igual ao valor de Valor.

e. Em Valor (terceira lista suspensa), digite ou selecione o valor que deseja especificar para os valores de Campo e Operador. 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 PHONE e TABLET.

f. Para adicionar outra regra, escolha Adicionar regra.

g. Para excluir uma regra, escolha o ícone X ao lado da 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.

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

a. Escolha Criar grupo de dispositivos estáticos.

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.

> **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)

- Execute o comando create-device-pool.

Para obter informações sobre como usar o Device Farm com o AWS CLI, consulteReferência do AWS CLI.

Criar um grupo de dispositivos (API)

- Chame a API `CreateDevicePool`.

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

Analisar os resultados 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.

O Device Farm também reúne outros artefatos, como arquivos, logs e imagens, que podem ser baixados quando a execução do teste for concluída.

Tópicos

- Trabalhar com relatórios de teste no Device Farm
- Trabalhando com artefatos no Device Farm

Trabalhar com relatórios de teste no Device Farm

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

Tópicos

Pré-requisitos

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

1. Para criar uma execução, consulte Criar 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 ter mais informações, consulte Compreensão dos resultados dos testes.

Compreensão dos resultados dos testes

O console do Device Farm exibe ícones que o ajudam a avaliar rapidamente o estado da execução de teste concluída.

Tópicos

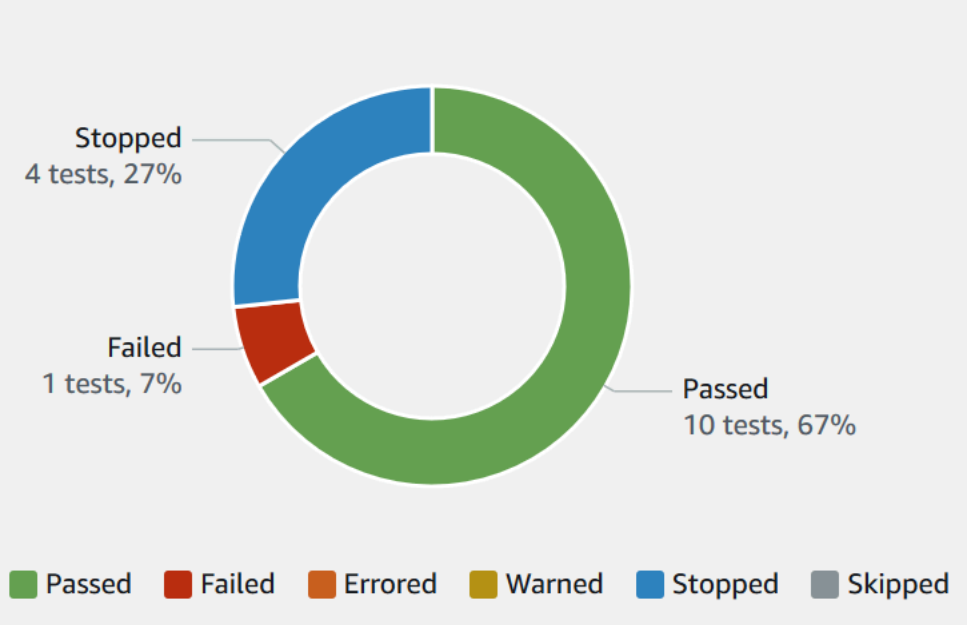
Relatório de resultados de um teste individual

Para relatórios que descrevem um teste individual, o Device Farm exibe um ícone:

Descrição	Ícone
O teste bem-sucedido.	
Falha no teste.	
O Device Farm pulou o teste.	
O teste parou.	
O Device Farm retornou um aviso.	
Device Farm retornou um erro.	

Relatórios de resultados de vários testes

Se você escolher uma execução concluída, o Device Farm exibirá um gráfico de resumo dos resultados do teste.



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

Os gráficos são sempre codificados por cores e rotulados.

Visualizar relatórios

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

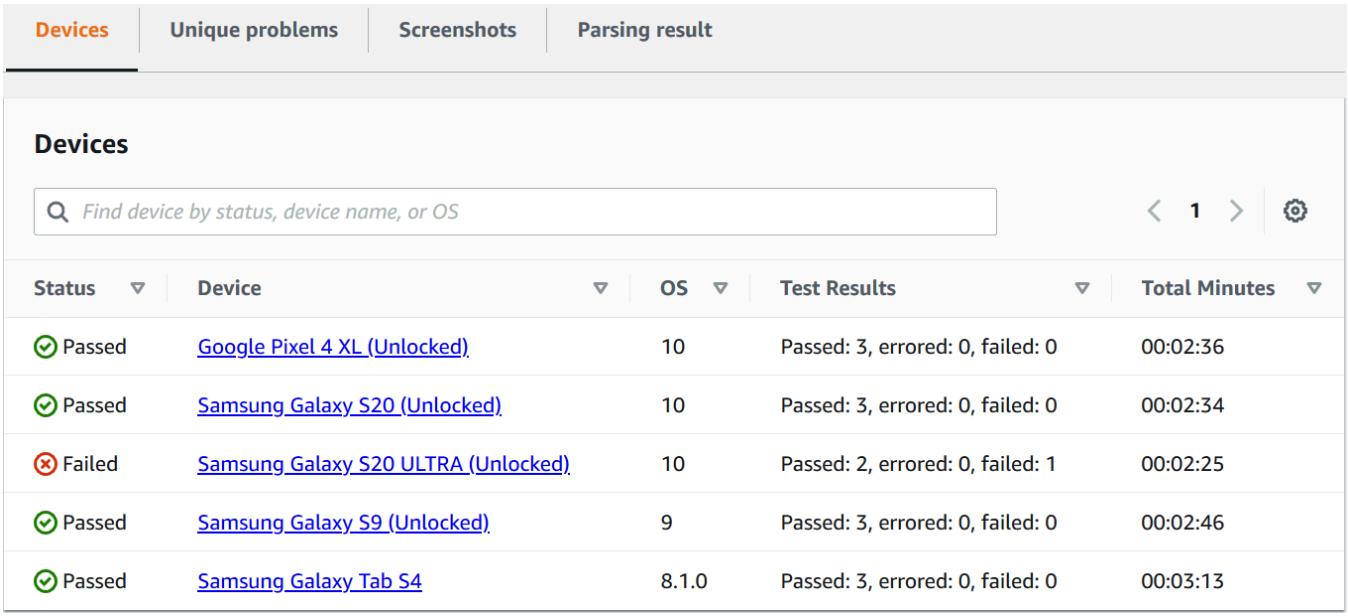
Tópicos

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 Mobile Device Testing e, em seguida, selecione 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 únicos lista avisos e falhas exclusivos. Para visualizar problemas exclusivos, siga as instruções em Visualizar relatórios com 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 02:36 minutos para serem executados.

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

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

Visualizar relatórios com 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 ter mais informações, consulte Relatório de resultados 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 Exibir informações de registro de um problema, dispositivo, suíte 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 capturou 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 suites 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 ter mais informações, consulte Relatórios de resultados de vários testes.

  Para visualizar os resultados completos por suíte, siga as instruções em Exibir 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 Exibir informações de registro de um problema, dispositivo, suíte 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 suítes 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 capturou durante a execução do dispositivo, agrupadas por suíte.

Exibir relatórios do conjunto de testes

1. Na seção Dispositivos, escolha o dispositivo.
2. Na seção Conjuntos, escolha a suíte 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 na suíte.

   Na tabela, a coluna Resultados do teste exibe o resultado. Esses dados também têm um componente gráfico. Para ter mais informações, consulte Relatórios de resultados 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 da suíte. Para visualizar essas informações, siga as instruções em Exibir informações de registro de um problema, dispositivo, suíte ou teste em um relatório.

   A seção Desempenho exibe informações sobre quaisquer dados de desempenho que o Device Farm tenha gerado 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 Pacotes, escolha o pacote.
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 ter mais informações, consulte Relatório de resultados 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 Exibir informações de registro de um problema, dispositivo, suíte 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 capturou durante o teste.

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

> **Note**
>
> No momento, o Device Farm coleta dados de desempenho do dispositivo somente para dispositivos Android.

A guia Desempenho exibe as seguintes informações:

- O gráfico da CPU exibe a porcentagem de CPU que o aplicativo 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 gráfico de memória exibe o número de MB que o aplicativo usou durante o problema, o dispositivo, o conjunto ou o 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, desde 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.

Exibir informações de registro de um problema, dispositivo, suíte 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:
  - O Harness representa uma entrada de registro que o Device Farm criou. Essas entradas de log normalmente são criadas durante os eventos de início e interrupção.
  - Dispositivo representa uma entrada de registro 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 do teste.

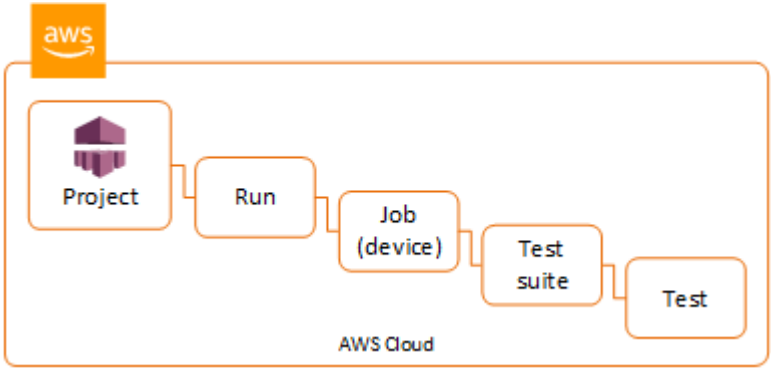
- Hora representa o tempo decorrido entre a primeira entrada de log e a entrada de log em questão. O tempo é expresso no formato *MM:SS.SSS*, em que *M* representa minutos e *S* representa segundos.
- PID representa o identificador de processo (PID) que criou a entrada de log. Todas as entradas de log 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 registro que correspondem a um valor de uma coluna específica, digite o valor na barra de pesquisa. Por exemplo, para mostrar todas as entradas de log com o valor Source de `Harness`, digite **`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 digitar ***** 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 Download de logs.

Trabalhando com artefatos no Device Farm



O Device Farm reúne artefatos como relatórios, arquivos de registro 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 ter mais informações, consulte Trabalhar com 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.

Usar 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 Uso de artefatos em um ambiente de teste personalizado.

Usar artefatos (AWS CLI)

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

Tópicos

Etapa 1: Obter os Amazon Resource Names (ARNs, Nomes de recurso da Amazon)

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

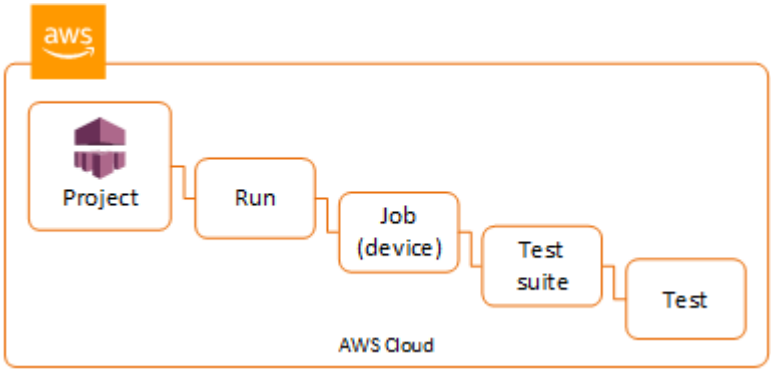
- Faça download do artefato usando o URL da etapa anterior. Este exemplo usa curl para fazer download de um arquivo de saída Android Appium Java:

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

Usar 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.

Uso de artefatos em um ambiente de teste personalizado



Em um ambiente de teste personalizado, o Device Farm reúne artefatos como relatórios personalizados, arquivos de registro 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 ter mais informações, consulte Trabalhando com artefatos no Device Farm.

Marcação de recursos do AWS Device Farm

O AWS Device Farm funciona com a API de marcação de grupos de recursos AWS. 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

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 utiliza uma lista de ARNs de serviços compatíveis

e uma lista de pares de chave-valor. O valor é opcional. Uma string vazia indica que não deve haver nenhum valor para essa tag. O exemplo de python a seguir marca uma série de ARNs de projeto 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 (`devicefarm:testgrid-session` resources) 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"])
```

Trabalho com tipos de teste no AWS Device Farm

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

Testar estruturas

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

Estruturas de teste de aplicativos Android

- Trabalhando com a Appium e o AWS Device Farm
- Trabalhar com instrumentação para Android e AWS Device Farm

Estruturas de testes do aplicativo iOS

- Trabalhando com a Appium e o AWS Device Farm
- Trabalhando com o XCTest para iOS e AWS Device Farm
- XCTest UI

Estruturas de testes do aplicativo web

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

Estruturas em um ambiente de teste personalizado

O Device Farm não é compatível com a personalização do ambiente de teste da estrutura XCTest. Para ter mais informações, consulte Trabalhar com ambientes de teste personalizados.

Suporte da versão do Appium

Para testes executados em um ambiente personalizado, o Device Farm é compatível com o Appium versão 1. Para ter mais informações, consulte Ambientes de teste.

Tipos de teste integrado

Com os testes incorporados, você pode testar seu aplicativo 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)

Trabalhando com a Appium e o 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 aplicativos da Web nativos 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.

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 seu 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 seu 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
```

Usando o `npm` comando para usar a versão mais recente do Appium 2, adicione esses comandos ao seu 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 CLI do AWS.

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

Tópicos

- Configure seu pacote de teste do Appium
- Criar um arquivo de pacote de teste compactado
- Faça o upload do seu pacote de teste no Device Farm
- Fazer capturas de tela de seus testes (Opcional)

Configure seu pacote de teste 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 para o diretório **<>**:

```
<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 para o diretório **<>**:

```
<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/
```

Confirme 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 a *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 de 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.

Faça o upload do seu pacote de teste 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 Mobile Device Testing 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 o projeto e escolha Criar uma nova execução.
5. Para testes nativos para Android e iOS

   Na página Escolher aplicativo, escolha Aplicativo móvel e selecione Escolher arquivo 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

   Na página Escolher aplicativo, selecione Web App.
6. Dê um nome apropriado ao seu teste. Ele pode conter qualquer combinação de espaços ou pontuação.
7. Escolha Próximo.
8. Na página Configurar, na seção Configurar estrutura de teste, escolha o ***idioma*** Appium e, em seguida, Escolher arquivo.

9. Procure e escolha o arquivo .zip que contém os testes. O arquivo .zip deve seguir o formato descrito em Configure seu pacote de teste do Appium.
10. Escolha Executar seu teste em um ambiente personalizado. Este ambiente de execução permite controle total sobre a configuração de teste, desmontagem e invocação, bem como a escolha de versões específicas de runtimes e do servidor Appium. Você pode configurar seu ambiente personalizado por meio do arquivo de especificação de teste. Para saber mais, consulte Trabalhar com ambientes de teste personalizados no AWS Device Farm.
11. Escolha Próxima etapa e siga as instruções para selecionar os dispositivos e iniciar a execução. Para ter mais informações, consulte Criar 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 em tempo de execução. 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.

Trabalho com testes do Android no AWS Device Farm

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

Estruturas de teste de aplicativos Android

Os testes personalizados estão disponíveis para dispositivos Android.

- Trabalhando com a Appium e o AWS Device Farm
- Trabalhar com instrumentação para Android e AWS Device Farm

Tipos de teste incorporados para Android

Há um tipo de teste integrado disponível para dispositivos Android.

- Integrado: Fuzz (Android e iOS)

Trabalhar com instrumentação para Android e AWS Device Farm

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

O Device Farm também fornece um aplicativo 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.

Tópicos

- O que é instrumentação?
- Upload dos testes de instrumentação para Android
- Fazer captura de telas em testes de instrumentação para Android
- Considerações adicionais para os testes de instrumentação para Android
- Análise de teste em modo padrão

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 Testes instrumentados na seção Tipos e locais de teste da documentação do Android Developer Tools.

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 Mobile Device Testing 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 Criar um projeto no AWS Device Farm.

4. Se o botão Criar uma nova execução for exibido, escolha-o.
5. Na página Escolher aplicativo, selecione Escolher arquivo.
6. Procure e escolha o arquivo de seu aplicativo Android. O arquivo deve ser .apk.
7. Escolha Próximo.
8. Na página Configurar, na seção Configurar estrutura de teste, escolha Instrumentação e, em seguida, selecione Escolher arquivo.
9. Procure e escolha o arquivo .apk que contém os testes.
10. Escolha Próximo e, em seguida, conclua as instruções restantes para selecionar dispositivos e iniciar a execução.

Fazer captura de telas em testes de instrumentação para 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`

Considerações adicionais para os testes de instrumentação para Android

Animações de sistema

De acordo com a documentação do Android para testes do Espresso, 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 android.support.test.runner.AndroidJ UnitRunner.

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 JUnit 4.

Para testar isso em um ambiente local:

1. Faça download `dex-test-parser` binário.
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
```

Trabalho com 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.

Estruturas de testes do aplicativo iOS

Os testes a seguir estão disponíveis para dispositivos iOS.

- Trabalhando com a Appium e o AWS Device Farm
- Trabalhando com o XCTest para iOS e AWS Device Farm
- 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)

Trabalhando com o XCTest para iOS e AWS Device Farm

Com o Device Farm, você pode usar a estrutura do XCTest para testar seu aplicativo em dispositivos reais. Para obter mais informações sobre o XCTest, consulte Testes básicos em Testar com Xcode.

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

Tópicos

- Criar os pacotes para a execução do XCTest
- Fazer upload dos pacotes para sua execução do XCTest no Device Farm

Criar os pacotes para a execução do XCTest

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

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

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 pacote do XCTest

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.

Fazer upload dos pacotes para sua execução do XCTest 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 Criar um projeto no AWS Device Farm.

   Caso contrário, no painel de navegação do Device Farm, escolha Mobile Device Testing e escolha Projetos.
3. Escolha o projeto que você deseja usar para executar o teste.
4. Escolha Criar uma nova execução.
5. Na página Escolher aplicativo, escolha Aplicativo móvel.
6. Selecione Escolher arquivo.
7. Procure o arquivo `.ipa` para seu aplicativo e faça o upload dele.

   > **Note**
   >
   > Seu pacote `.ipa` deve ser compilado para testes.
8. Após a conclusão do upload, escolha Próximo.
9. Na página Configurar, na seção Configuração da estrutura de teste, escolha XCTest. Selecione Escolher arquivo.
10. Procure o arquivo `.zip` que contém o pacote do XCTest para seu aplicativo e faça o upload dele.
11. Após a conclusão do upload, escolha Próximo.
12. Conclua as etapas restantes no processo de criação do projeto. Selecione os dispositivos que deseja testar e especifique o estado do dispositivo.
13. Depois de configurar sua execução, na página Revisar e iniciar a execução, escolha Confirmar e iniciar a execução.

    O Device Farm executa o teste e mostra os resultados no console.

Trabalhar com a estrutura de testes de interface do usuário XCTest para iOS e AWS Device Farm

O Device Farm oferece suporte para a estrutura de teste da interface do usuário do XCTest para iOS. Especificamente, o Device Farm é compatível com os testes de interface do usuário do XCTest escritos em Objective-C e Swift.

Tópicos

O que é a estrutura de teste XCTest UI?

A estrutura XCTest UI é a nova estrutura de teste introduzida no Xcode 7. Essa estrutura XCTest UI amplia o XCTest com recursos de testes de IU. Para obter mais informações, consulte Teste da interface do usuário na biblioteca de desenvolvedor do iOS.

Preparação dos testes do XCTest UI para iOS

O pacote do executor de testes XCTest UI para iOS deve estar contido em um arquivo .ipa devidamente formatado.

Para criar um arquivo.ipa, coloque seu pacote my-project-name UITest-runner.app em um diretório vazio do Payload. Em seguida, arquive o diretório Payload em um arquivo .zip e mude a extensão do arquivo para .ipa. O pacote *UITest-Runner.app é produzido pelo Xcode quando você compila o projeto para testes. Ele pode ser encontrado no diretório Products do projeto.

Upload dos testes do XCTest UI para 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 Mobile Device Testing 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 Criar um projeto no AWS Device Farm.

4. Se o botão Criar uma nova execução for exibido, escolha-o.
5. Na página Escolher aplicativo, selecione Escolher arquivo.
6. 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.

7. Escolha Próximo.
8. Na página Configurar, na seção Configurar estrutura de teste, escolha XCTest UI e selecione Escolher arquivo.
9. Procure e selecione o arquivo .ipa que contém o executor de testes do XCTest UI para iOS.
10. Escolha Próximo e, em seguida, conclua as instruções restantes para selecionar os dispositivos em que os testes serão executados e iniciar a execução.

Fazer capturas de tela em testes do XCTest UI para iOS

Os testes do XCTest UI fazem capturas de tela automaticamente para todas as etapas de teste. Essas capturas de tela são exibidas no seu relatório de teste do Device Farm. Nenhuma outro código é necessário.

Trabalhando com testes de aplicativos web no AWS Device Farm

O Device Farm fornece testes com o Appium para aplicativos da web. Para obter mais informações sobre como configurar seus testes Appium no Device Farm, consulte. the section called “Appium”

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 Preços 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.

Trabalhando com testes integrados no AWS Device Farm

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

Tipos de teste integrado

Os testes integrados possibilitam testar os aplicativos sem escrever scripts.

- Integrado: Fuzz (Android e iOS)

Trabalhar com o teste de fuzz incorporado para o Device Farm

O Device Farm oferece um tipo de teste de fuzz incorporado.

O que é o teste de fuzz incorporado?

O teste integrado Fuzz envia aleatoriamente eventos de interface de usuário para os dispositivos e, em seguida, relata os resultados.

Usar o tipo de teste integrado Fuzz

Use o console do Device Farm para executar o teste de fuzz incorporado.

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

2. No painel de navegação do Device Farm, escolha Mobile Device Testing 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 Criar um projeto no AWS Device Farm.

4. Se o botão Criar uma nova execução for exibido, escolha-o.
5. Na página Escolher aplicativo, selecione Escolher arquivo.
6. Procure e escolha o arquivo de aplicativo no qual você deseja executar o teste integrado Fuzz.
7. Escolha Próximo.
8. Na página Configurar, na seção Configuração da estrutura de teste, escolha Integrado: Fuzz.
9. 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.
   - Limite de eventos: especifique um número entre 1 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.

10. Escolha Próximo e, em seguida, conclua as instruções restantes para selecionar dispositivos e iniciar a execução.

Trabalhar com 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 Ambientes de teste.

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.
- Registro ao vivo e streaming de vídeo: seus registros 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 Carregando uma especificação de teste personalizada usando e. AWS CLI Criar uma execução de teste no Device Farm

Tópicos

- Sintaxe da especificação de teste

- Exemplo da especificação de teste
- Trabalhando com o ambiente de teste Amazon Linux 2 para testes de Android
- Variáveis de ambiente
- Migrar testes de um ambiente de teste padrão para um ambiente de teste personalizado
- Extensão de ambientes de teste personalizados no Device Farm

Sintaxe da especificação de teste

Esta é a estrutura do arquivo de especificação de teste YAML:

```
version: 0.1

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

artifacts:
  - location
  - location
```

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 registro e imagens de um local especificado aqui. Os caracteres curinga não são suportados 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 Uso de 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 da especificação de teste

Este é um exemplo de uma especificação de teste YAML do Device Farm que configura uma execução de teste do 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
```

Trabalhando com o 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 do framework 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

Software compatível

O host de teste do Amazon Linux 2 vem pré-instalado com muitas das bibliotecas de software necessárias para suportar 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 a partir do seu pacote de teste, fazer o download da Internet ou acessar fontes privadas na sua VPC (consulte VPC ENI para obter mais informações). Para obter mais informações, consulte o exemplo do arquivo de especificação de teste.

As seguintes versões de software estão atualmente disponíveis no host:

Software Library	Software Version	Command to use in your test spec file
Python	3.8	`devicefarm-cli usa python 3.8`
	3.9	`devicefarm-cli usa python 3.9`
	3.10	`devicefarm-cli usa python 3.10`
Java	8	`devicefarm-cli usa java 8`
	11	`devicefarm-cli usa java 11`
	17	`devicefarm-cli usa java 17`
NodeJS	16	`devicefarm-cli usa o nó 16`
	18	`devicefarm-cli usa o nó 16`

Ruby	2.7	`devicefarm-cli usa ruby 2.7`
	3.2	`devicefarm-cli usa ruby 3.2`
Appium	1	`devicefarm-cli usa appium 1`
	2	`devicefarm-cli usa appium 1`

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

A ferramenta **`devicefarm-cli`**

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 `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 `help` página de `devicefarm-cli`:

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

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

Vamos analisar alguns exemplos usando `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, `devicefarm-cli` também alterna as ferramentas de suporte para essas linguagens, como `pip` para Python e `npm` para NodeJS.

Seleção de host de teste 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 `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 `preview_features` sinalizadores e substituí-la pelo novo `android_test_host` campo para evitar a sobrecarga de manutenção no futuro.

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

Exemplo de arquivo de especificações de teste

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
```

Migrar para o Amazon Linux 2 Test Host

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 uma nova execução de teste no projeto.
4. Escolha um aplicativo e um pacote de teste usados anteriormente para sua estrutura de teste.
5. Escolha Executar seu 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 a 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, certifique-se de que, ao usar o `devicefarm-cli` para selecionar suas versões de software, você selecione as versões que estão na lista de versões suportadas.
- 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

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 em tempo de execução.

Tópicos

Variáveis de ambiente comuns

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 `ANDROID_HOME` ambiente 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 ambiente do Appium Java JUnit.

Esta seção descreve variáveis de ambiente usadas pelos testes do Appium Java JUnit em um ambiente de teste personalizado.

$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 do TestNG

Esta seção descreve variáveis de ambiente usadas pelos testes do Appium Java TestNG em um ambiente de teste personalizado.

$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 do XCUITest

$DEVICEFARM_XCUITESTRUN_FILE

Caminho para o `.xctestun` arquivo 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.

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

O guia a seguir explica como alternar de um modo de execução de teste padrão para um modo de execução personalizado. 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 levantar e mudar e alcançar a paridade com o ambiente local e streaming de vídeo ao vivo.

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 registros, 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 em sua (s) máquina (s) local (s). 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: os modos de execução padrão e personalizado fornecem vídeos e registros para seus testes. No entanto, no modo padrão, você obtém o vídeo e os registros 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 registros de seus testes no lado do cliente. Além disso, você pode baixar o vídeo e outros artefatos ao final do (s) teste (s).

4. Suspensão de uso: os seguintes tipos de teste serão descontinuados até o final de dezembro de 2023 no modo de execução padrão:

   - Appium (todos os idiomas)

- Calabash
- XCTest
- UI Automation
- UI Automator
- Testes na Web
- Integrado ao Explorer

Depois de descontinuado, você não poderá usar essas estruturas no modo padrão. Em vez disso, você pode usar o modo personalizado para os tipos de teste listados acima.

> **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 do seu aplicativo (ou selecione`web app`), escolha o tipo de estrutura de teste, faça o upload do pacote de teste e, abaixo do `Choose your execution environment` parâmetro, 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 de `testSpecArn` parâmetros da `ScheduleRun` API 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 a fim de mover os testes iOS XCUITest para um ambiente de teste personalizado.

Extensão de ambientes de teste personalizados no Device Farm

O Device Farm Custom Mode 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.

Configurando um PIN

Alguns aplicativos exigem que você defina um PIN no dispositivo. O Device Farm não suporta 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..

Acelerar os testes baseados no Appium 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 do seu aplicativo quando uma sessão do Appium termina, efetivamente não fazendo nenhuma limpeza. `fullReset` desinstala e limpa todos os dados do aplicativo do dispositivo após o encerramento da sessão. Para obter mais informações, consulte Estratégias de redefinição na documentação do Appium.

`ignoreUnimportantViews` (Somente Android)

Instrui a 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 baseados em XPath 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 a Appium a assumir que uma dependência essencial do iOS,`webDriverAgent`, já está em execução e disponível para aceitar solicitações HTTP na URL especificada. Se ainda `webDriverAgent` não estiver instalado e funcionando, o Appium pode levar algum tempo no início de uma suíte de testes para iniciar o. `webDriverAgent` Se você iniciar `webDriverAgent` sozinho e configurar o Appium `webDriverAgentUrl` `http://localhost:8100` ao iniciar o Appium, poderá inicializar sua suíte de testes mais rapidamente. Observe que esse recurso nunca deve ser usado junto com o `useNewWDA` recurso.

Você pode usar o código a seguir para começar com seu arquivo `webDriverAgent` de especificação de teste na porta local do dispositivo e, em seguida`8100`, encaminhá-lo para a porta local do host de teste `8100` (isso permite que você defina `webDriverAgentUrl` o valor como`http://localhost:8100`). Esse código deve ser executado durante a fase de instalação após a definição de qualquer código para configurar o Appium e as variáveis de `webDriverAgent` ambiente:

```
      # Start WebDriverAgent and iProxy
      - >-
        xcodebuild test-without-building -project /usr/local/avm/versions/
$APPIUM_VERSION/node_modules/appium/node_modules/appium-webdriveragent/
WebDriverAgent.xcodeproj
        -scheme WebDriverAgentRunner -derivedDataPath
 $DEVICEFARM_WDA_DERIVED_DATA_PATH
        -destination id=$DEVICEFARM_DEVICE_UDID_FOR_APPIUM
 IPHONEOS_DEPLOYMENT_TARGET=$DEVICEFARM_DEVICE_OS_VERSION
        GCC_TREAT_WARNINGS_AS_ERRORS=0 COMPILER_INDEX_STORE_ENABLE=NO >>
 $DEVICEFARM_LOG_DIR/webdriveragent_log.txt 2>&1 &

        iproxy 8100 8100 >> $DEVICEFARM_LOG_DIR/iproxy_log.txt 2>&1 &
```

Em seguida, você pode adicionar o código a seguir ao seu arquivo de especificação de teste para garantir que ele tenha sido `webDriverAgent` 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 que o Appium suporta, consulte Appium Desired Capabilities na documentação do Appium.

Usando webhooks e outras APIs após a execução dos testes

Você pode fazer com que o Device Farm chame um webhook depois que cada suíte 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 Enviar sua primeira mensagem do Slack usando o Webhook na referência da API do Slack.

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 para outras 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 das APIs de terceiros 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 Solicitação de credenciais de segurança temporárias no Guia do Usuário do IAM.

Adicionar arquivos extras ao seu pacote de teste

Talvez você queira usar arquivos adicionais como parte de seus testes como arquivos extras de configuração ou dados de teste adicionais. Você pode adicionar esses arquivos adicionais ao seu pacote de teste antes de carregá-lo 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 que suportam 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 todos os arquivos extras em um local ligeiramente diferente devido à forma como os pacotes de teste do iOS são AWS Device Farm resignados. Ao criar seu teste para 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 pacotes do XCUITest, adicione qualquer arquivo extra ao diretório que termina em.app dentro do diretório Payload. 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 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 variável de ambiente *$DEVICEFARM_TEST_PACKAGE_PATH*. (Isso significa que se você adicionasse um arquivo chamado

extra_file à raiz do arquivo, como no primeiro exemplo, ele estaria localizado em *$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 arquivo *testng.xml* em 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 diretório */tmp*:

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

No caso de um arquivo APK ou JAR, você encontraria seus arquivos extras descompactados no diretório /tmp (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 diretório *Payload*. *Por exemplo, com base no exemplo de IPA acima, o arquivo seria encontrado no local /tmp/payload/adfiosreferenceappuitests-runner.app/extra_file (referenciável como /tmp/payload/ *.app/extra_file).*

Trabalhando com 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 registros 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.

Tópicos

Crie uma sessão de acesso remoto no AWS Device Farm

Para obter informações sobre sessões de acesso remoto, consulte Sessões.

Pré-requisitos

- Crie um projeto no Device Farm. Siga as instruções em Criar 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 Mobile Device Testing e, em seguida, escolha Projects.
3. Se você já tiver um projeto, selecione na lista. Caso contrário, crie um projeto seguindo as instruções em Criar 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:
   - Nome
   - 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 Device requested (Solicitado pelo dispositivo) 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.

Use 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

- Crie uma sessão. Siga as instruções em Crie 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 um aplicativo no dispositivo de sessão, em Instalar aplicativos, selecione Escolher arquivo e, em seguida, 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 aplicativos executados no dispositivo escolhendo Aplicativos 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 Obter resultados da sessão.

Dicas e truques

É possível que você tenha problemas de desempenho com a sessão de acesso remoto em algumas regiões da AWS. 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.

Obter resultados de uma sessão de acesso remoto no AWS Device Farm

Para obter informações sobre sessões, consulte Sessões.

Pré-requisitos

- Conclua uma sessão. Siga as instruções em Use 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 Mobile Device Testing e, em seguida, escolha Projects.
2. Escolha o projeto que contém a sessão.
3. Escolha Acesso remoto e, em seguida, 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.

Trabalhando com 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. 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 ter mais informações, consulte Gerenciamento de dispositivos privados 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 aplicativos privados aos 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 ter mais informações, consulte Uso dos serviços de endpoint do Amazon VPC com o Device Farm.

Se você estiver interessado em usar a frota de um ou mais dispositivos privados, entre em contato conosco. A equipe do Device Farm deve trabalhar com você para configurar e implantar uma frota de dispositivos privados AWS em sua conta.

Tópicos

Gerenciamento de 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. Esse dispositivo é exclusivo da sua conta AWS.

> **Note**
>
> Atualmente, os dispositivos privados estão disponíveis apenas na região AWS Oeste dos EUA (Oregon) (`us-west-2`).

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.

Este tópico explica como criar um perfil de instância e executar outras tarefas de gerenciamento de dispositivos comuns.

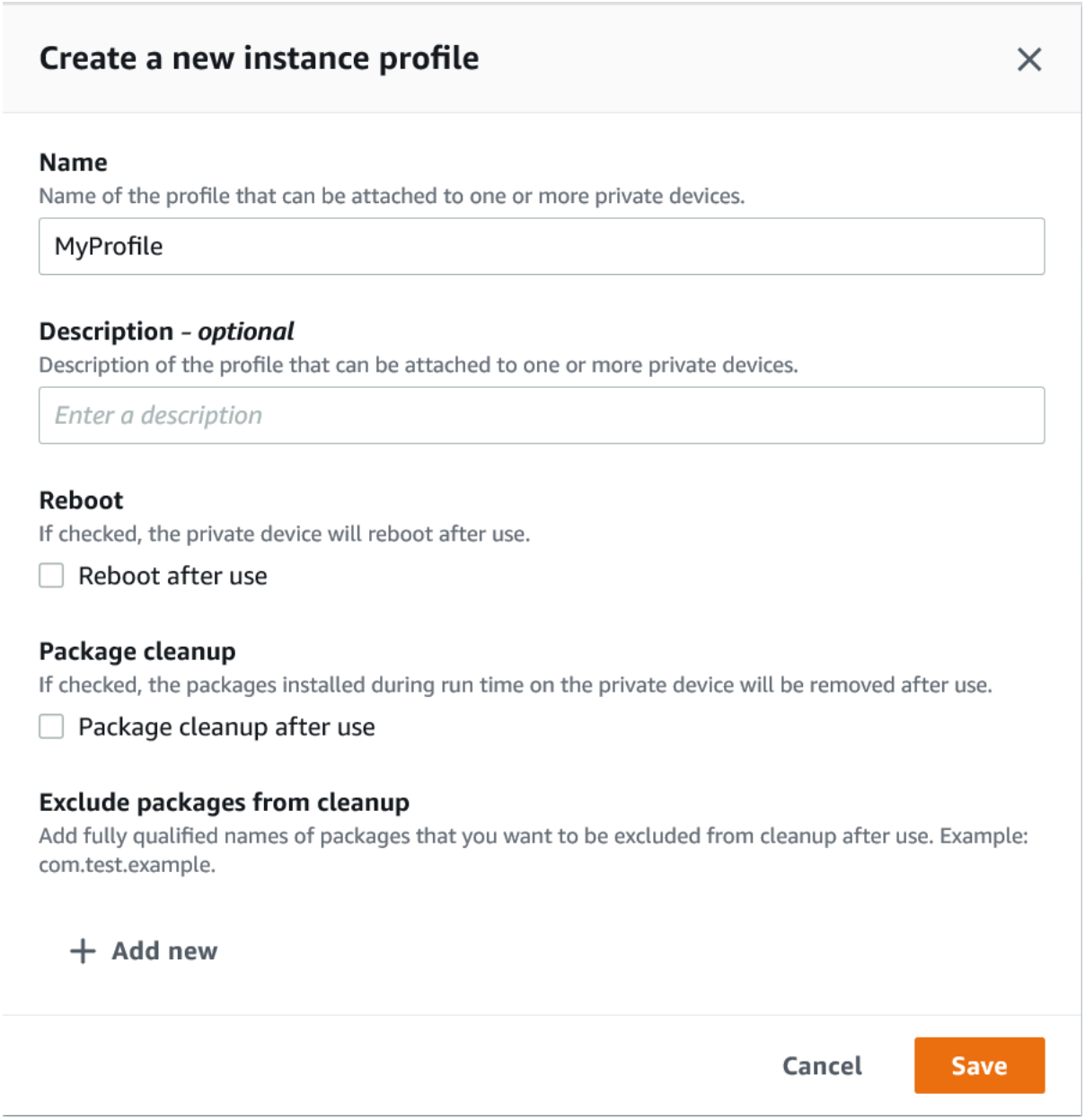
Tópicos

Criar um perfil da instância

Para controlar o comportamento dos dispositivos privados durante uma execução de teste ou sessão de acesso remoto, você pode criar ou modificar um perfil de instância no Device Farm. Você não precisa de um perfil de instância para começar a usar os dispositivos privados.

1. Abra o console do Device Farm em https://console.aws.amazon.com/devicefarm/.
2. No painel de navegação do Device Farm, escolha Teste de dispositivos móveis e, em seguida, escolha 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:

   - Reiniciar após o uso - Para reiniciar o dispositivo, marque essa caixa de seleção. Por padrão, essa caixa de seleção fica desmarcada (`false`).

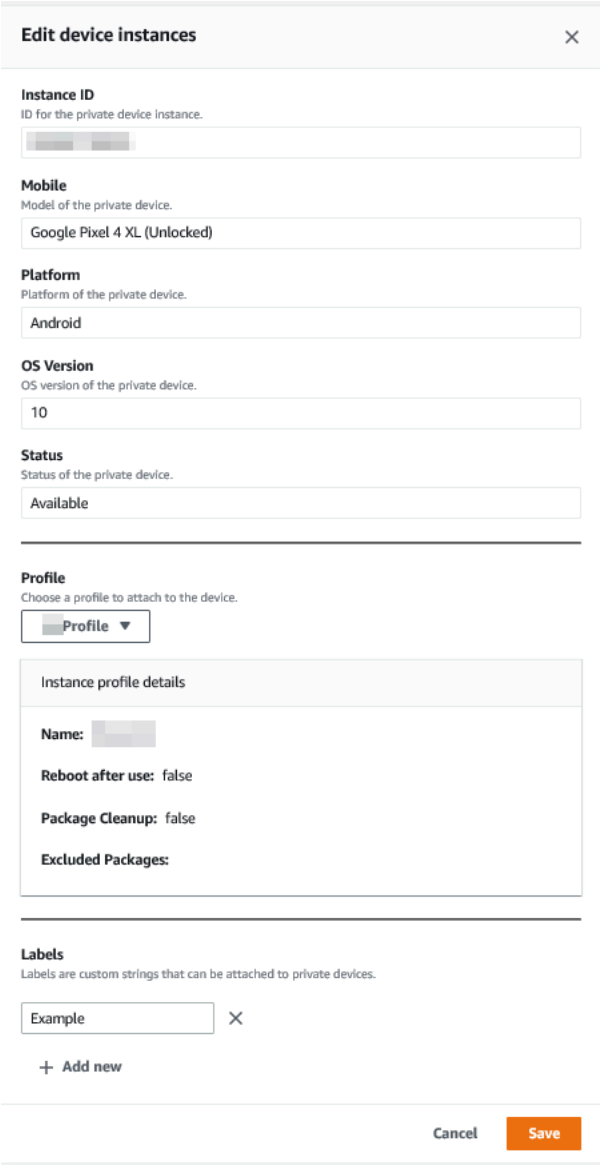
- Limpeza de pacotes Para remover todos os pacotes de aplicativos que você instalou no dispositivo, marque essa caixa de seleção. Por padrão, essa caixa de seleção fica desmarcada (`false`). Para manter todos os pacotes de aplicativos que você instalou no dispositivo, deixe esta caixa de seleção desmarcada.
- Excluir Pacotes de Limpeza - Para manter apenas pacotes selecionados de aplicativos no dispositivo, marque a caixa de seleção Limpeza do pacote e, em seguida, 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 aplicativos no dispositivo, escolha Adicionar novo e, em seguida, digite o nome totalmente qualificado de cada pacote.

8. Escolha Salvar.

Gerenciar uma instância de dispositivo privado

Se já tiver um ou mais dispositivos privados em sua frota, você poderá visualizar informações a respeito e gerenciar determinadas configurações para cada instância do dispositivo. Você também pode solicitar uma instância de dispositivo privado adicional.

1. Abra o console do Device Farm em https://console.aws.amazon.com/devicefarm/.
2. No painel de navegação do Device Farm, escolha Teste de dispositivos móveis e, em seguida, escolha 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. (Opcional) 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, em seguida, selecione Editar.

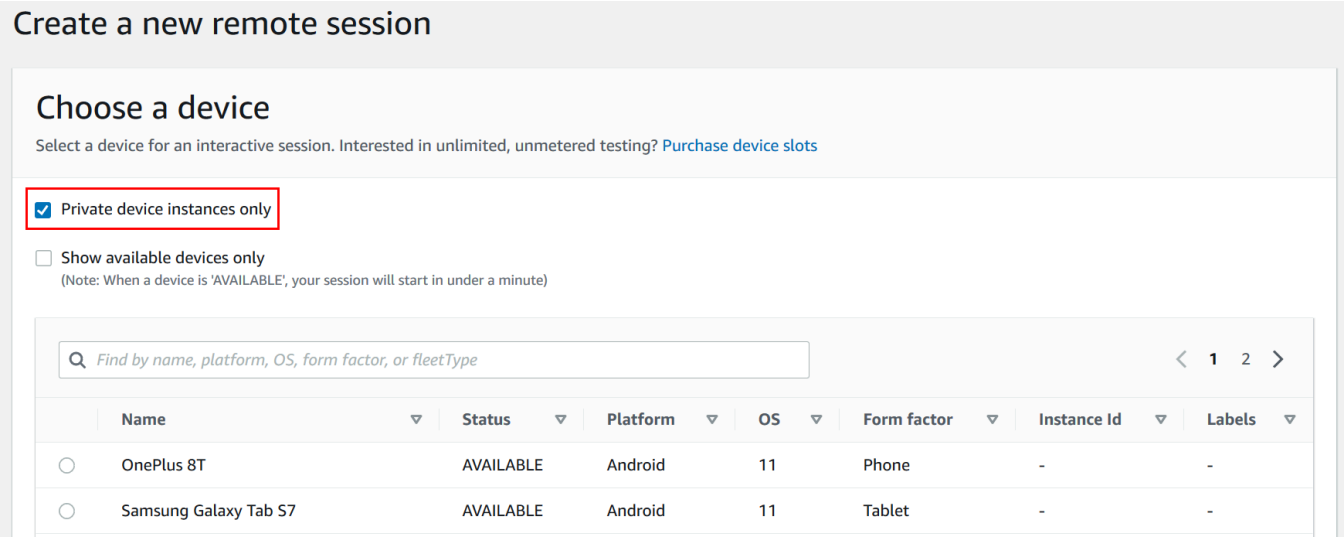


6. (Opcional) Em Perfil, escolha um perfil de instância para anexar a instância do dispositivo. Isso pode ser útil se você deseja sempre excluir um pacote de aplicativo específico das tarefas de limpeza, por exemplo.
7. (Opcional) Em Rótulos, escolha Adicionar novo para adicionar um rótulo à instância 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

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.

1. Abra o console do Device Farm em https://console.aws.amazon.com/devicefarm/.
2. No painel de navegação do Device Farm, escolha Mobile Device Testing e, em seguida, escolha Projects.
3. Escolha um projeto existente na lista ou crie um novo. Para criar um novo projeto, selecione Novo projeto, digite um nome para o projeto e, em seguida, selecione Enviar.
4. Faça um dos seguintes 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 apenas os dispositivos privados que a equipe do Device Farm configurou e associou à sua conta AWS. 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 Escolher um dispositivo, selecione Somente instâncias de dispositivos privados para limitar a lista apenas aos dispositivos privados que a equipe do Device Farm configurou e associou à sua conta AWS. 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.



Próximas etapas

Depois de configurar os dispositivos privados, você também pode gerenciar seus dispositivos privados das seguintes formas:

- Ignorar a assinatura nova em dispositivos privados
- Use os serviços de endpoint do Amazon Virtual Private Cloud com o Device Farm

Para excluir um perfil de instância, no menu Perfis de instância, escolha a opção de alternância ao lado da instância que você deseja excluir e, em seguida, escolha Excluir.

Seleção de dispositivos privados em um pool de dispositivos

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 (AWSCLI) ou a API

Device Farm para criar ou modificar um pool de dispositivos com dispositivos privados usando essas regras.

Tópicos

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 dispositivos `Apple iPhone 13` no iOS versão `16.1.0` em sua frota particular, 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 Gerenciamento de dispositivos privados.

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 dispositivos `iPhone 13` no OS `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 ARNs do dispositivo e os rótulos da instância do dispositivo não se adequarem ao seu caso de uso. Os ARNs de instância de dispositivo geralmente são usados como regras para grupos de dispositivos quando uma instância específica de dispositivo é configurada de forma única e específica como 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 de ARNs de instância de dispositivos para seu pool de dispositivos, precisará alterar manualmente a definição de regra do seu 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 aos ARNs do dispositivo). 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.

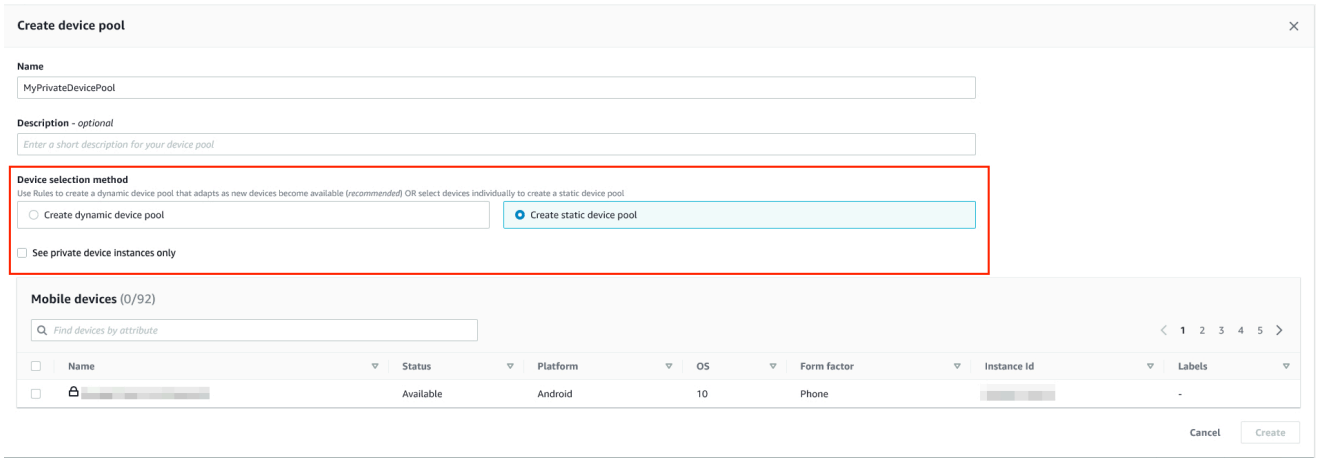
Criação de um pool de dispositivos privados com dispositivos privados (console)

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.

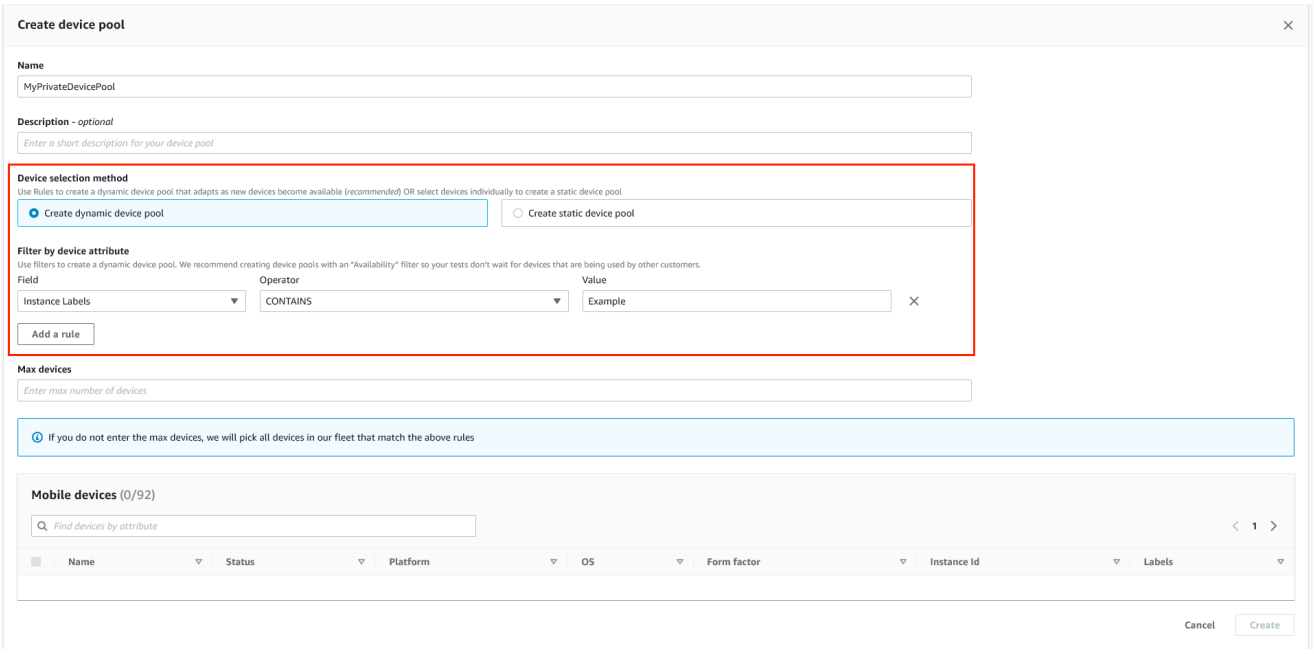
> **Note**
>
> Ao criar um pool de dispositivos com dispositivos privados no console, você só pode usar qualquer uma das três regras disponíveis para selecionar dispositivos privados. Se você quiser criar um pool de dispositivos que contenha vários tipos de regras para dispositivos privados (por exemplo, pools de dispositivos que contêm regras para ARNs de dispositivos e ARNs de instâncias de dispositivos), você precisa 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 Mobile Device Testing e, em seguida, escolha Projects.
3. Escolha um projeto existente na lista ou crie um novo. Para criar um novo projeto, selecione Novo projeto, digite um nome para o projeto e, em seguida, selecione Enviar.
4. Escolha Testes automatizados, em seguida, Criar nova execução. O assistente orienta você durante as etapas para escolher seu aplicativo e configurar o teste que você deseja executar.
5. Na etapa Selecionar dispositivos, escolha Criar um novo grupo de dispositivos e insira um nome e uma descrição opcional para o grupo de dispositivos.

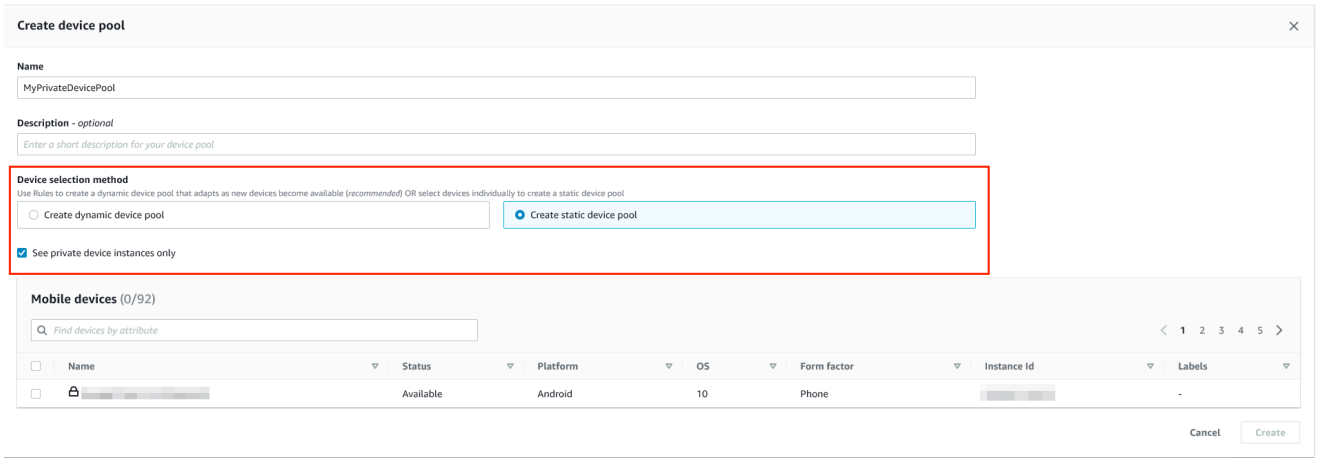
    a. Para usar regras de ARN de dispositivos para seu pool de dispositivos, escolha Criar pool estático de dispositivos e selecione os tipos de dispositivos específicos da lista que você gostaria de usar no pool de dispositivos. Não selecione Instâncias de dispositivos privados somente porque essa opção faz com que o pool de dispositivos seja criado com regras de ARN de instância de dispositivo (em vez de regras de ARN de dispositivo).



b. Para usar regras de rótulo de instância de dispositivo para seu pool de dispositivos, escolha Criar pool dinâmico de dispositivos. Em seguida, para cada etiqueta que você gostaria de usar no pool de dispositivos, escolha Adicionar uma regra. Para cada regra, escolha Rótulos de instância como o`Field`, escolha Contém como `Operator` o. e especifique o rótulo de instância do dispositivo desejado como `Value` o.



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 pool 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, consulte Referência do AWS CLI.

Criação de um pool de dispositivos privados com dispositivos privados (API)

- Chame a API `CreateDevicePool`.

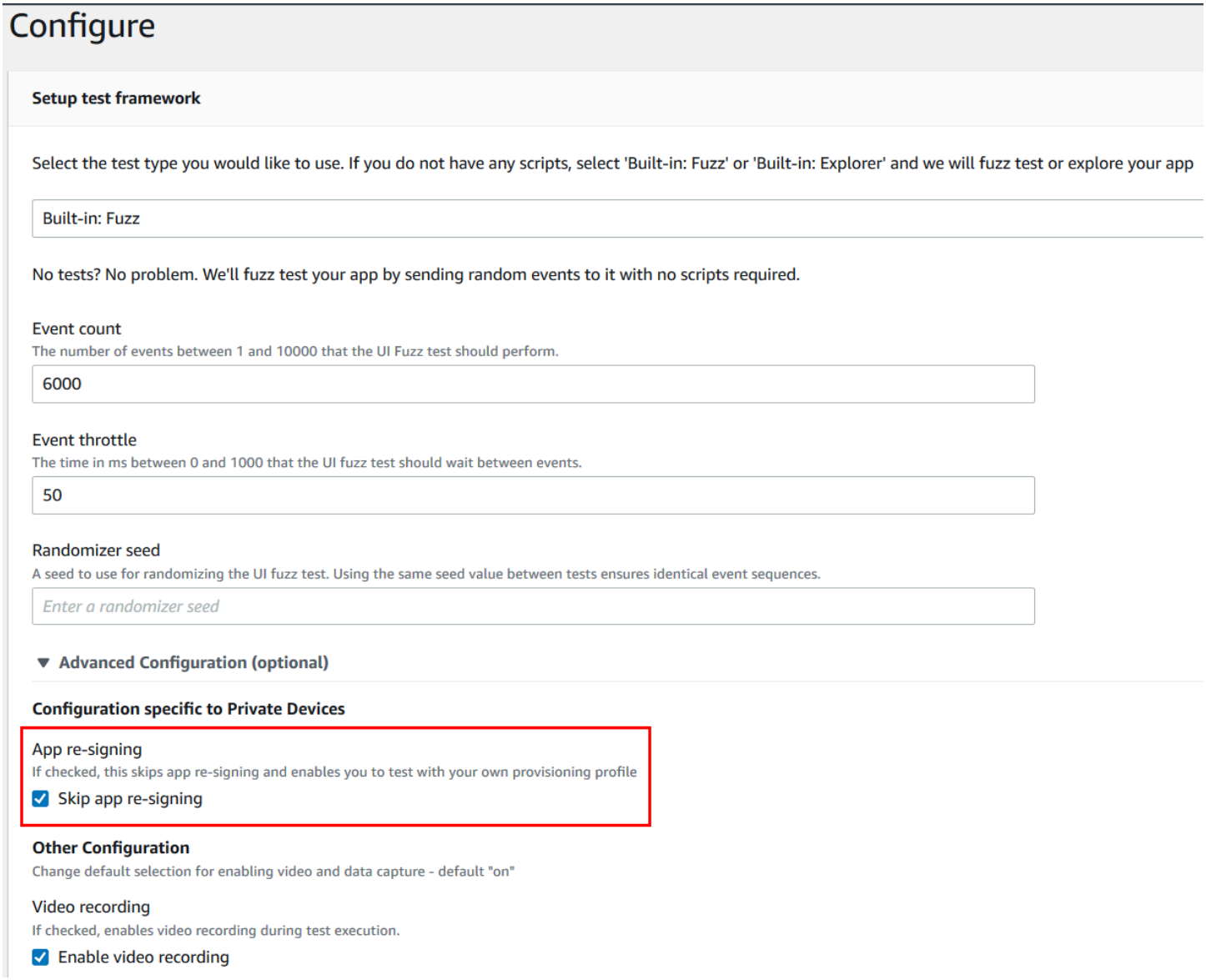
Para obter informações sobre como usar a API do Device Farm, consulte Automatização do Device Farm .

Ignorar a nova assinatura de aplicativos em dispositivos privados no AWS Device Farm

Quando você usa dispositivos privados, pode ignorar a etapa em que o AWS Device Farm assina novamente seu aplicativo. Isso é diferente dos dispositivos públicos, em que o Device Farm sempre assina novamente seu aplicativo 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 pode ser útil se o seu aplicativo tiver uma funcionalidade que seja interrompida quando o Device Farm assinar novamente o aplicativo. 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 as Perguntas frequentes sobre o AWS Device Farm.

Para ignorar a nova assinatura do aplicativo para a execução de um teste, selecione Skip app re-signing (Ignorar nova assinatura do aplicativo) na página Configure (Configurar) ao criar a execução de teste.



> **Note**
>
> Se você estiver usando a estrutura XCTest, a opção Skip app re-signing (Ignorar nova assinatura do aplicativo) não estará disponível. Para obter mais informações, consulte Trabalhando com o XCTest para iOS e AWS Device Farm.

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 do aplicativo 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 do aplicativo 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 no seu aplicativo”.

- 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 seu aplicativo assinado novamente em .

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 Certificates, IDs, and Profiles (Certificados, IDs 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 `ListDeviceInstances` operação da API 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 no seu aplicativo iOS

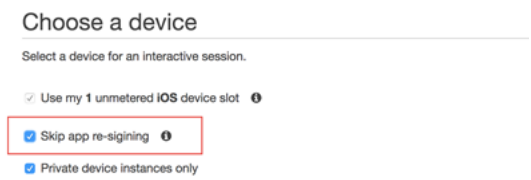
Se estiver usando um perfil de provisionamento de desenvolvedor interno (Enterprise), você precisará fazer um único procedimento para confiar no certificado de desenvolvedor do aplicativo interno em cada um dos dispositivos privados.

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 Criar um perfil da instância para criar ou modificar um perfil de instância no Device Farm. No perfil da instância, adicione o ID de pacote do aplicativo de teste ou aplicativo fictício à configuração Excluir pacotes da limpeza. Em seguida, anexe o perfil da instância à instância do dispositivo privado para garantir que o Device Farm não remova esse aplicativo 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 Crie 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, certifique-se de adicionar o aplicativo fictício ou o aplicativo que você deseja testar à configuração Exclude Packages from Cleanup (Excluir pacotes da limpeza) para o perfil da instância que está anexado a essa instância.
2. Quando a sessão remota for iniciada, escolha Escolher arquivo para instalar um aplicativo 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.

Uso dos serviços de endpoint do Amazon VPC com o Device Farm

> **Note**
>
> O uso do Amazon VPC Endpoint Services com o Device Farm só é suportado para clientes com dispositivos privados configurados. Para habilitar sua conta AWS para usar esse recurso com dispositivos privados, entre em contato conosco.

O Amazon Virtual Private Cloud (Amazon VPC) é um serviço da AWS que você pode usar para iniciar recursos da AWS 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 usar o Amazon VPC para hospedar aplicativos privados na região oeste dos EUA (Oregon) (`us-west-2`) da AWS, poderá estabelecer uma conexão privada entre seu VPC e o Device Farm. Com essa conexão, você pode usar o Device Farm para testar aplicativos privados sem expô-los à Internet pública. Para habilitar sua conta AWS para usar esse recurso com dispositivos privados, entre em contato conosco.

Para conectar um recurso em seu VPC ao Device Farm, você pode usar o console do Amazon VPC para criar um serviço de endpoint VPC. Esse serviço de endpoint permite que você forneça o recurso em sua VPC ao Device Farm por meio de um endpoint VPC do Device Farm. O serviço de endpoint fornece conectividade confiável e dimensionável ao Device Farm sem exigir um gateway de Internet, uma instância de tradução de endereços de rede (NAT) ou uma conexão VPN. Para obter mais informações, consulte Serviços de endpoint VPC (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 da AWS PrivateLink. Embora a conexão seja segura e privada, essa segurança depende da proteção de suas credenciais da AWS. Se as suas credenciais da AWS forem comprometidas, um invasor poderá acessar ou expor os dados do seu serviço ao mundo externo.

Depois de criar um serviço de ponto de extremidade VPC no Amazon VPC, você pode usar o console do Device Farm para criar uma configuração de ponto de extremidade VPC no Device Farm. Este tópico mostra como criar a conexão do Amazon VPC e a configuração do ponto de extremidade do VPC no Device Farm.

Antes de começar

As informações a seguir são para usuários do 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 VPC com os quais você pode usá-lo. Quando você criar e configurar um serviço de endpoint VPC para trabalhar com o Device Farm, certifique-se de escolher 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 balanceador de carga de rede associado a um serviço de ponto de extremidade de VPC determina as zonas de disponibilidade para esse serviço de ponto de extremidade de VPC. Se o serviço de endpoint da VPC não mostrar todas as três zonas de disponibilidade, você deverá recriar o balanceador de carga da rede para ativar essas três zonas e, em seguida, associar novamente o balanceador de carga da rede ao serviço de endpoint.
- As entidades permitidas para o serviço de endpoint devem incluir o Amazon Resource Name (ARN) do endpoint VPC do Device Farm (ARN do serviço). Depois de criar seu serviço de endpoint, adicione o ARN do serviço de endpoint VPC do Device Farm à sua lista de permissões para dar permissão ao Device Farm para acessar seu serviço de endpoint VPC. Para obter o ARN do serviço de VPC endpoint do , entre em contato conosco.

Além disso, se você mantiver a configuração Aceitação obrigatória ativada ao criar seu serviço de endpoint VPC, deverá aceitar manualmente cada solicitação de conexão que o Device Farm enviar ao serviço de endpoint. Para alterar essa configuração de um serviço de ponto de extremidade existente, escolha o serviço de ponto de extremidade no console do Amazon VPC, selecione Ações e, em seguida, selecione Modificar configuração de aceitação de ponto de extremidade. Para obter mais informações, consulte Alterar os balanceadores de carga e as configurações de aceitação no Guia AWS PrivateLink.

A próxima seção explica como criar um serviço de endpoint do 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 balanceador de carga de rede 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. Escolha Criar um balanceador de carga.
4. Em Network Load Balancer, escolha Criar.
5. Na página Criar balanceador de carga de rede, em Configuração básica, faça o seguinte:
   a. Insira um Nome de balanceador de carga.
   b. Em Esquema, escolha Interno.
6. Em Mapeamento de rede, faça o seguinte:
   a. Escolha a VPC para seu grupo-alvo.
   b. Selecione os seguintes Mapeamentos:
      - `us-west-2a`
      - `us-west-2b`
      - `us-west-2c`
7. Em Ouvintes e roteamento, use as opções Protocolo e Porta para escolher o grupo-alvo.

   > **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 alvos sejam registrados em cada uma dessas Zonas de Disponibilidade ou, se você registrar alvos em menos de todas as três zonas, exige que você ative o balanceamento de carga entre zonas. Caso contrário, o balanceador de carga pode não funcionar como esperado.

8. Escolha Criar um balanceador de carga.

Old console

Para criar um balanceador de carga de rede 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. Escolha 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 de balanceador de carga.
   b. Em Esquema, escolha Interno.
6. Em Ouvintes, selecione o Protocolo e a Porta que o grupo-alvo está usando.
7. Em Zonas de disponibilidade, faça o seguinte:
   a. Escolha a VPC para seu grupo-alvo.
   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 alvos sejam registrados em cada uma dessas Zonas de Disponibilidade ou, se você registrar alvos em menos de todas as três zonas, exige que você ative o 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: criação de um serviço de endpoint do Amazon VPC

Depois de criar o Network Load Balancer, use o console do Amazon VPC para criar um serviço de endpoint no seu 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. Faça um dos seguintes procedimentos:

   - Se você já tiver um balanceador de carga de rede que deseja que o serviço de endpoint use, escolha-o em Balanceadores de carga disponíveis e continue na etapa 5.
   - Se você ainda não tiver criado um Balanceador de carga de rede, escolha Criar novo balanceador de carga. O console do Amazon EC2 é aberto. Siga as etapas em Criar um balanceador de carga de rede, começando pela etapa 3, e continue com estas etapas no console do 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 diretores.
9. Entre em contato conosco para obter o ARN do endpoint VPC do Device Farm (ARN do serviço) a ser adicionado à lista de permissões do serviço de endpoint e, em seguida, 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.

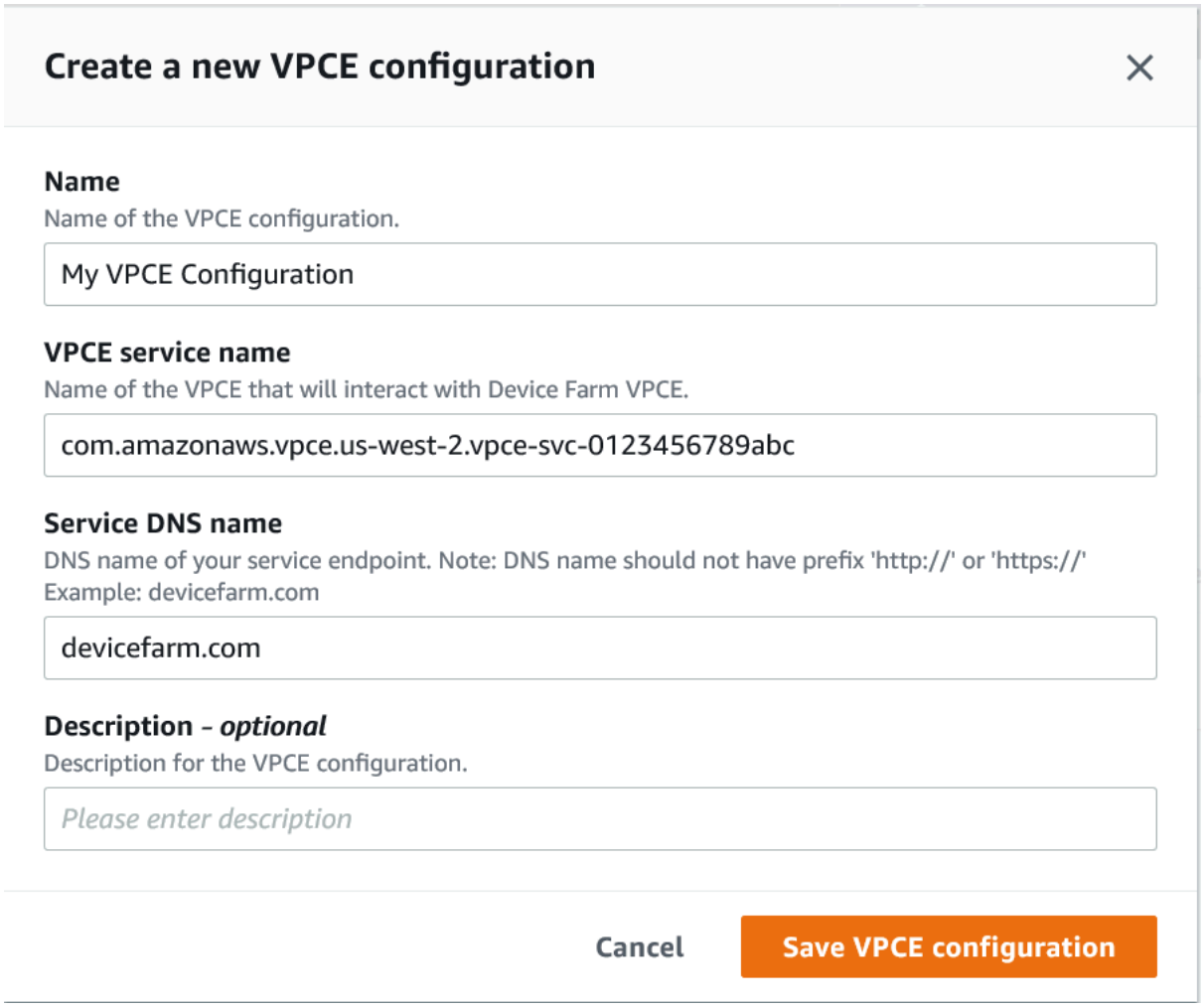
Seu serviço de endpoint VPC agora está pronto para ser usado com o Device Farm.

Etapa 3: Criação de uma configuração de ponto de extremidade VPC no Device Farm

Depois de criar um serviço de endpoint no Amazon VPC, você pode criar uma configuração de endpoint do 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 de dispositivo móvel e, em seguida, Dispositivos privados.
3. Escolha Configurações de VPCE.
4. Escolha Criar uma configuração de VPCE.
5. Em Criar uma nova configuração de VPCE, digite um Nome para a configuração do endpoint VPC.
6. Para o nome do serviço VPCE, digite o nome do serviço de endpoint do Amazon VPC (nome do serviço) que você anotou no console do 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 o aplicativo 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 seu serviço de endpoint 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 ponto de extremidade da VPC, você pode usar a configuração para criar execuções de teste ou sessões de acesso remoto. Para ter mais informações, consulte Criar uma execução de teste no Device Farm ou Crie uma sessão..

Trabalhar com o Amazon VPC em regiões da AWS

Os serviços do Device Farm estão localizados apenas na região oeste dos EUA (Oregon)(`us-west-2`). Você pode usar o Amazon Virtual Private Cloud (Amazon VPC) para acessar um serviço em seu Amazon Virtual Private Cloud em outra região da AWS usando o Device Farm. Se o Device Farm e seu serviço estiverem na mesma região, consulte Uso dos serviços de endpoint do Amazon VPC com o Device Farm.

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 O que é emparelhamento de VPC? no Guia de emparelhamento do Amazon VPC.

emparelhamento de VPC

Você pode colocar duas VPCs em pares em regiões diferentes, desde que elas tenham blocos CIDR distintos e não sobrepostos. Isso garante que todos os endereços IP privados sejam exclusivos e permite que todos os recursos nas VPCs se enderecem uns 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.

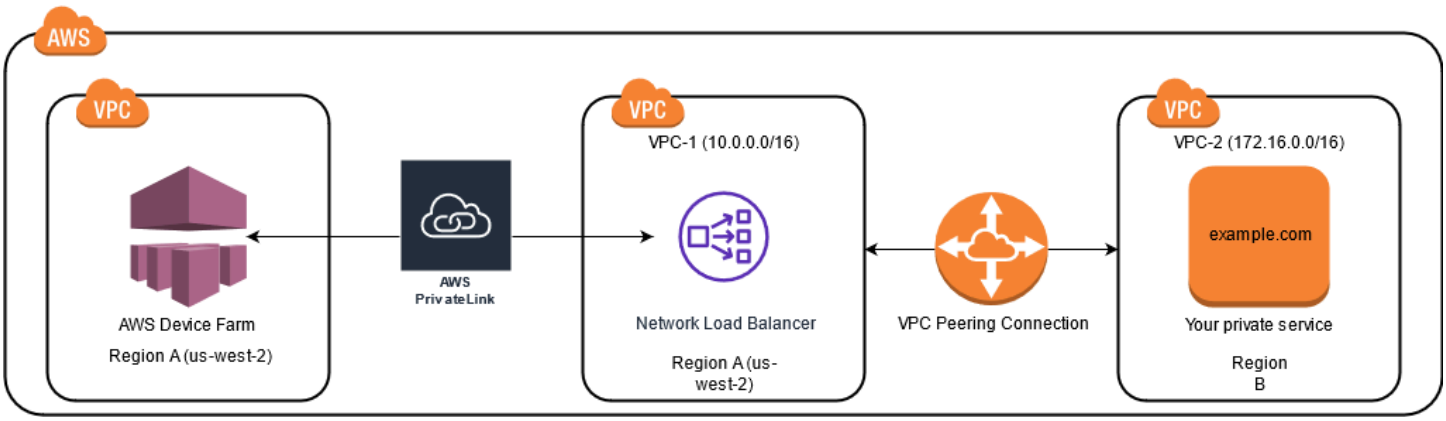
Este tópico inclui um cenário de exemplo entre regiões no qual o Device Farm (denominado VPC-1) está localizado na região Oeste dos EUA (Oregon) (`us-west-2`). A segunda VPC neste exemplo (chamada de VPC-2) está em outra região.

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**
>
> O estabelecimento de uma conexão de peering entre duas VPCs pode alterar a postura de segurança das VPCs. Além disso, a adição de novas entradas em suas tabelas de rotas pode alterar a postura de segurança dos recursos dentro das 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

Este exemplo requer o seguinte:

- Duas VPCs configuradas com sub-redes que contêm 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 a criação de VPCs e a configuração de sub-redes, consulte Trabalho com VPCs e sub-redes no Guia de emparelhamento do Amazon VPC.

Etapa 1: configurar uma conexão de emparelhamento entre a VPC-1 e a VPC-2

Estabeleça uma conexão de peering entre as duas VPCs que contenham blocos CIDR não sobrepostos. Para fazer isso, consulte Criar e aceitar conexões de emparelhamento VPC no Guia de emparelhamento do Amazon VPC. Usando o cenário entre regiões deste tópico e o Guia de emparelhamento do 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 Cotas do Amazon VPC no Guia de emparelhamento do Amazon VPC.

Etapa 2: atualizar as tabelas de rotas na VPC-1 e na VPC-2

Depois de configurar uma conexão de emparelhamento, você deve estabelecer uma rota de destino entre as duas VPCs para transferir dados entre elas. 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 Atualizar suas tabelas de rotas para uma conexão de emparelhamento VPC no Guia de emparelhamento VPC da Amazon. Usando o cenário entre regiões deste tópico e o Guia de emparelhamento do 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 balanceador de carga de rede 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 balanceador de carga de rede) e fazer com que esse recurso estático encaminhe as solicitações para o seu verdadeiro destino.

     > **Note**
     > - Se estiver direcionando uma ou mais instâncias autônomas do Amazon Elastic Compute Cloud (Amazon EC2), abra o console do Amazon EC2 em https://console.aws.amazon.com/ec2/ e escolha Instâncias.
     > - Se você estiver direcionando um grupo de instâncias do Amazon EC2 Auto Scaling, deverá associar o grupo do Amazon EC2 Auto Scaling a um balanceador de carga de

rede. Para obter mais informações, consulte Anexar um balanceador de carga ao seu grupo Auto Scaling no Guia do Usuário do Amazon EC2 Auto Scaling.

Em seguida, abra o console do Amazon EC2 em https://console.aws.amazon.com/ec2/ e escolha 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 o endereço IPv4, escolha o endereço IP do 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 para Balanceadores de Carga de Rede, 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
```

Endereço IPv4

```
172.16.100.60
```

Portas

```
80
```

Etapa 4: Criar um balanceador de carga de rede

Crie um balanceador de carga de rede 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
```

Scheme

```
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 ponto de extremidade VPC

Você pode usar o Network Load Balancer para criar um serviço de endpoint VPC. Por meio desse serviço de endpoint VPC, o Device Farm pode se conectar ao seu serviço no VPC-2 sem nenhuma infraestrutura adicional, como um gateway de Internet, uma instância NAT ou uma conexão VPN.

Para fazer isso, consulte Criação de um serviço de endpoint do Amazon VPC.

Etapa 6: criar uma configuração de ponto de extremidade VPC no 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 Criação de uma configuração de ponto de extremidade VPC no 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

Você pode criar execuções de teste que usam a configuração de endpoint VPC descrita na etapa 6. Para ter mais informações, consulte Criar uma execução de teste no Device Farm ou Crie uma sessão..

Crie uma rede dimensionável com o Transit Gateway

Para criar uma rede dimensionável usando mais de duas VPCs, você pode usar o Transit Gateway para atuar como um hub de trânsito de rede para interconectar suas VPCs e redes locais. 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 O que é um transit gateway? no Guia de Transit Gateways do Amazon VPC.

Encerramento de dispositivos privados

> **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.

```
<Para encerrar um dispositivo privado após o prazo inicial acordado, você deve f
```

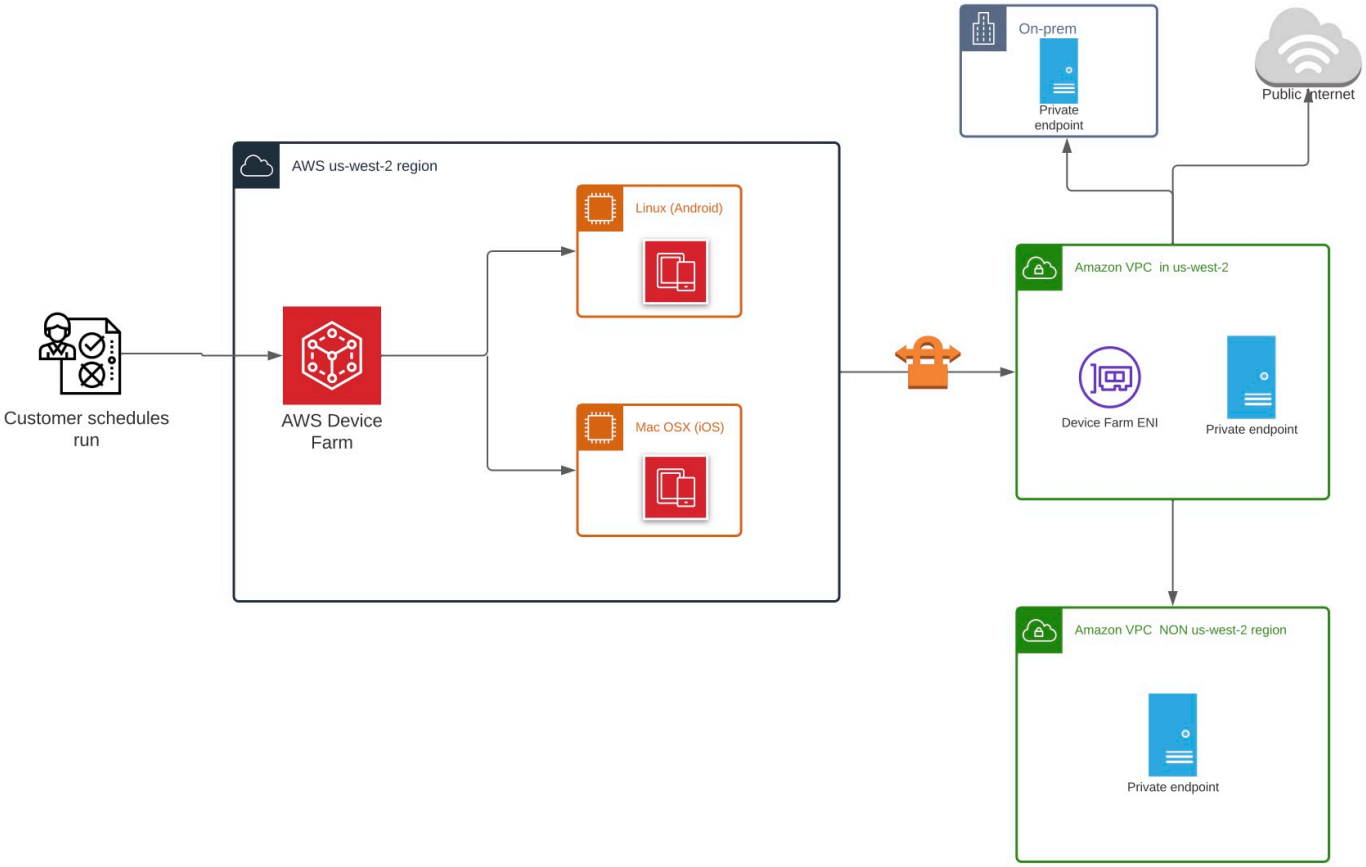
VPC-ENI no AWS Device Farm

> **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 dispositivos móveis 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 serviços e aplicativos isolados e não voltados para a Internet por meio de uma interface de rede elástica. Para obter mais informações sobre VPCs, consulte o Guia do usuário do Amazon VPC.

Se seu endpoint privado ou VPC não estiver na região, você poderá vinculá-lo `us-west-2` a uma VPC na `us-west-2` região usando soluções como Transit Gateway ou VPC Peering. Em tais situações, o Device Farm criará uma ENI em uma sub-rede que você fornece para a `us-west-2` VPC da sua região, e você será responsável por garantir que uma conexão possa ser estabelecida entre a VPC da região e a `us-west-2` VPC na outra região.



Para obter informações sobre como usar AWS CloudFormation para criar e emparelhar VPCs automaticamente, consulte os modelos de VPCpeering no repositório de modelos AWS CloudFormation no GitHub.

> **Note**
>
> O Device Farm não cobra nada pela criação de ENIs na `us-west-2` VPC de um cliente em. 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, CDNs públicas), a menos

que haja um gateway NAT especificado na VPC. Para obter mais informações, consulte Gateways NAT no Guia do usuário da Amazon VPC.

Tópicos

AWScontrole de acesso e IAM

O AWS Device Farm permite que você use AWS Identity and Access Management(IAM) para criar políticas concedendo ou restringindo 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 da função vinculada ao serviço:

```
"iam:CreateServiceLinkedRole"
```

> **Note**
>
> Nenhuma dessas permissões é necessária para usuários que não usam configurações de VPC em seus projetos.

Funções vinculadas ao serviço

O AWS Device Farm usa AWS Identity and Access Management funções vinculadas a serviços (IAM). Uma função vinculada ao serviço é um tipo exclusivo de função IAM vinculada diretamente ao Device Farm. As funções vinculadas ao serviço são predefinidas pelo Device Farm e incluem todas as permissões que o serviço requer para chamar outros serviços AWS em seu nome.

Uma função vinculada a um 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 às funções vinculadas ao serviço, consulte Serviços da AWS que funcionam com o IAM e procure os serviços com Sim na coluna Função vinculada ao serviço. Escolha um Sim com um link para exibir a documentação da função vinculada a serviço desse serviço.

Permissões de função vinculadas ao serviço para o Device Farm

O Device Farm usa a função vinculada ao serviço denominada AWSServiceRoleForDeviceFarm — Permite que o Device Farm acesse recursos da AWS da em seu nome.

A função vinculada ao serviço AWSServiceRoleForDeviceFarm 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
  - Descrever VPCs
  - Descrever sub-redes
  - Descrever grupos de segurança
  - Excluir interfaces
  - Modificar interfaces de rede
- Para interfaces de rede
  - Criar tags
- Para interfaces de rede do EC2 gerenciadas pela Device Farm
  - Criar permissões de interface de rede

A política completa do IAM 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 uma função vinculada ao serviço. Para obter mais informações, consulte Permissões de função vinculada ao serviço no Guia do usuário do IAM.

Criação de uma função vinculada ao serviço para o Device Farm

Quando você fornece uma configuração de VPC para um projeto de teste móvel, você não precisa criar manualmente uma função vinculada ao serviço. Quando você cria seu primeiro recurso do Device Farm naAWS Management Console, na ou na AWS APIAWS CLI, o Device Farm cria o perfil vinculado ao serviço para você.

Se excluir essa função vinculada ao serviço e precisar criá-la novamente, você poderá usar esse mesmo processo para recriar a função em sua conta. Quando você cria seu primeiro recurso do Device Farm, o Device Farm cria a função vinculada ao serviço para você novamente.

Você também pode usar o console IAM para criar uma função vinculada a um serviço com o caso de uso Device Farm. Na AWS CLI ou na API do AWS, crie uma função vinculada ao serviço com o nome de serviço `devicefarm.amazonaws.com`. 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 uma função vinculada ao serviço para o Device Farm

O Device Farm não permite que você edite a função vinculada ao serviço AWSServiceRoleForDeviceFarm. Depois que criar uma função vinculada ao serviço, você não poderá alterar o nome da função, pois várias entidades podem fazer referência a ela. No entanto, será possível editar a descrição da função usando o IAM. Para obter mais informações, consulte Editar uma função vinculada ao serviço no Guia do usuário do IAM.

Excluir uma função vinculada ao serviço para o Device Farm

Se você não precisar mais usar um recurso 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 sua função vinculada ao serviço antes de excluí-la 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 a função vinculada ao serviço usando o IAM

Use o console IAM, a API AWS CLI ou AWS para excluir a função vinculada ao serviço AWSServiceRoleForDeviceFarm. Para obter mais informações, consulte Excluir uma função vinculada ao serviço no Guia do usuário do IAM.

Regiões compatíveis com as funções vinculadas ao serviço Device Farm

O Device Farm suporta o uso de funções vinculadas a serviços em todas as regiões em que o serviço está disponível. Para obter mais informações, consulte Regiões e endpoints do AWS.

O Device Farm não suporta o uso de funções vinculadas a serviços em todas as regiões em que o serviço está disponível. Você pode usar a função AWSServiceRoleForDeviceFarm nas seguintes regiões.

Nome da região	Identidade da região	Support em Device Farm
Leste dos EUA (N. 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
Asia Pacific (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 do 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 a VPCs somente na `us-west-2` região. Se você não tiver uma VPC na `us-west-2` região, precisará criar uma. Em seguida, 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 o emparelhamento de VPCs, consulte o Guia de emparelhamento do Amazon VPC.

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) do Device Farm.

- A resolução de DNS é necessária na VPC que você usa.
- Depois que sua VPC for criada, você precisará das seguintes informações sobre a VPC na região: `us-west-2`
  - ID da VPC
  - IDs de sub-rede
  - IDs de grupos de segurança
- Você deve configurar as conexões do 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 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 do Amazon VPC. A configuração da VPC-ENI é configurada para cada projeto. Um projeto só pode ter um endpoint VPC-ENI por vez. Para configurar o acesso da VPC a um projeto, você deve conhecer os seguintes detalhes:

- A ID da VPC, `us-west-2` se seu aplicativo estiver hospedado lá, ou a ID da `us-west-2` VPC 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 Mobile Device Testing e, em seguida, escolha Projects.
3. Em Projetos de teste móvel, escolha o nome do seu projeto na lista.
4. Escolha Project settings (Configurações do projeto).
5. Na seção Configurações da Virtual Private Cloud (VPC), você pode alterar oVPC, e. `Subnets` `Security Groups`
6. Escolha Save (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 IPv4 disponíveis.
- 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 tradução 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.
- Somente o tráfego de saída da ENI gerenciada pelo serviço é suportado. Isso significa que a ENI não pode receber solicitações de entrada não solicitadas da VPC.

Registro de chamadas de API do AWS Device Farm com AWS CloudTrail

O AWS Device Farm é integrado ao AWS CloudTrail, um serviço que fornece um registro das ações realizadas por um usuário, função ou serviço da AWS no AWS Device Farm. O 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 ativar o fornecimento contínuo de eventos do CloudTrail para um bucket do Amazon S3, incluindo eventos para o AWS Device Farm. Se você não configurar uma trilha, ainda poderá visualizar os eventos mais recentes no console do CloudTrail em Histórico de eventos. Usando as informações coletadas pelo CloudTrail, você pode determinar a solicitação que foi feita ao AWS Device Farm, o endereço IP do qual a solicitação foi feita, quem fez a solicitação, quando foi feita e outros detalhes.

Para saber mais sobre o CloudTrail, consulte o Guia do usuário do AWS CloudTrail.

Informações do AWS Device Farm no CloudTrail

O CloudTrail é habilitado em sua conta AWS quando ela é criada. Quando ocorre uma atividade no AWS Device Farm, essa atividade é registrada em um evento do CloudTrail junto com outros eventos de serviço AWS no Histórico de eventos. Você pode visualizar, pesquisar e baixar eventos recentes em sua conta da AWS. Para obter mais informações, consulte Visualização de eventos com o histórico de eventos do CloudTrail.

Para obter um registro contínuo de eventos na sua conta AWS, incluindo eventos para o AWS Device Farm, crie uma trilha. Uma trilha permite que o CloudTrail entregue arquivos de log a 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 partição da AWS e entrega os arquivos de registro no bucket do Amazon S3 que você especificou Além disso, é possível configurar outros serviços da AWS para analisar ainda mais e agir com base nos dados de eventos coletados nos logs do CloudTrail. Para obter mais informações, consulte:

- Visão geral da criação de uma trilha
- Serviços e integrações compatíveis com o CloudTrail
- Configurar notificações do Amazon SNS para o CloudTrail

- Receber arquivos de log do CloudTrail de várias regiões e receber arquivos de log do CloudTrail de várias contas

Quando o registro do CloudTrail está ativado na sua conta AWS, as chamadas de API feitas para as ações do Device Farm são rastreadas nos arquivos de registro. Os registros do Device Farm são gravados junto com outros registros de serviço AWS em um arquivo de log. O CloudTrail determina quando criar e gravar em um novo arquivo de acordo com o período e o tamanho do arquivo.

Todas as ações do Device Farm são registradas e documentadas no Referência do AWS CLI e no Automatização do Device Farm. Por exemplo, as chamadas para criar um novo projeto ou executar no Device Farm geram entradas nos arquivos de registro do CloudTrail.

Cada entrada de log ou evento contém informações sobre quem gerou a solicitação. As informações de identidade ajudam a determinar:

- Se a solicitação foi feita com credenciais de usuário raiz ou do 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 serviço da AWS.

Para obter mais informações, consulte o Elemento userIdentity do CloudTrail.

Compreensão das entradas do arquivo de registro do AWS Device Farm

Uma trilha é uma configuração que permite a entrega de eventos como arquivos de log a um bucket do Amazon S3 especificado. Os arquivos de log do CloudTrail 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 de solicitação e assim por diante. Os arquivos de log do CloudTrail não são um rastreamento de pilha ordenada de chamadas de API pública. Dessa forma, eles não são exibidos em uma ordem específica.

O exemplo a seguir mostra uma entrada de registro do CloudTrail que demonstra a ação Device Farm `ListRuns`:

```
{
```

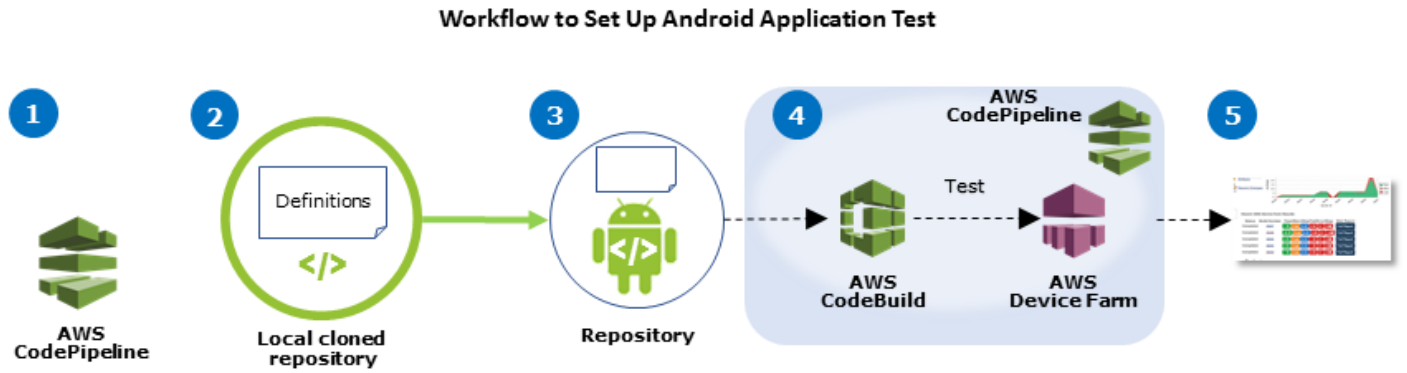
```
  "Records": [
    {
      "eventVersion": "1.03",
      "userIdentity": {
        "type": "Root",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:root",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
          "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 ...
          ]
        }
      }
    }
  ]
}
```

Uso do AWS Device Farm em um estágio de teste do CodePipeline

Você pode usar o AWS CodePipeline para incorporar testes de aplicativos móveis configurados 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 a configuração desse pipeline, consulte o Tutorial: Criar e testar um aplicativo Android quando enviado ao 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

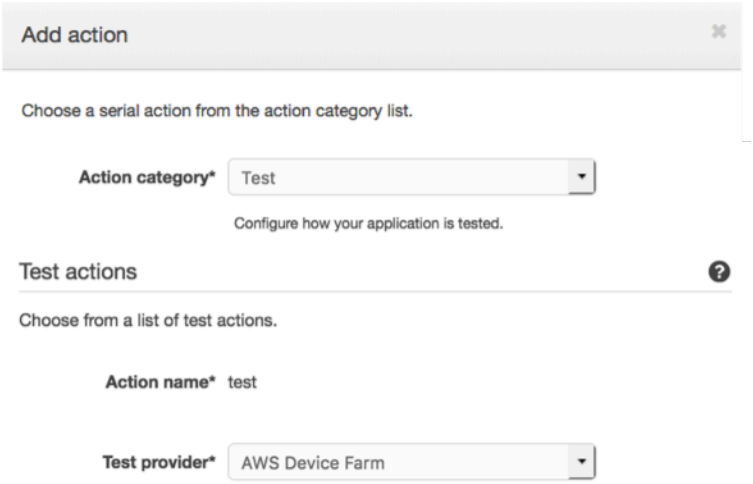
Para saber como configurar um pipeline que testa continuamente um aplicativo compilado (como um arquivo iOS `.ipa` ou arquivo Android `.apk`) como sua origem, consulte Tutorial: Testar um aplicativo iOS toda vez que você carregar um arquivo .ipa em um bucket do Amazon S3.

Configure o CodePipeline para usar seus testes do Device Farm

Nestas 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.

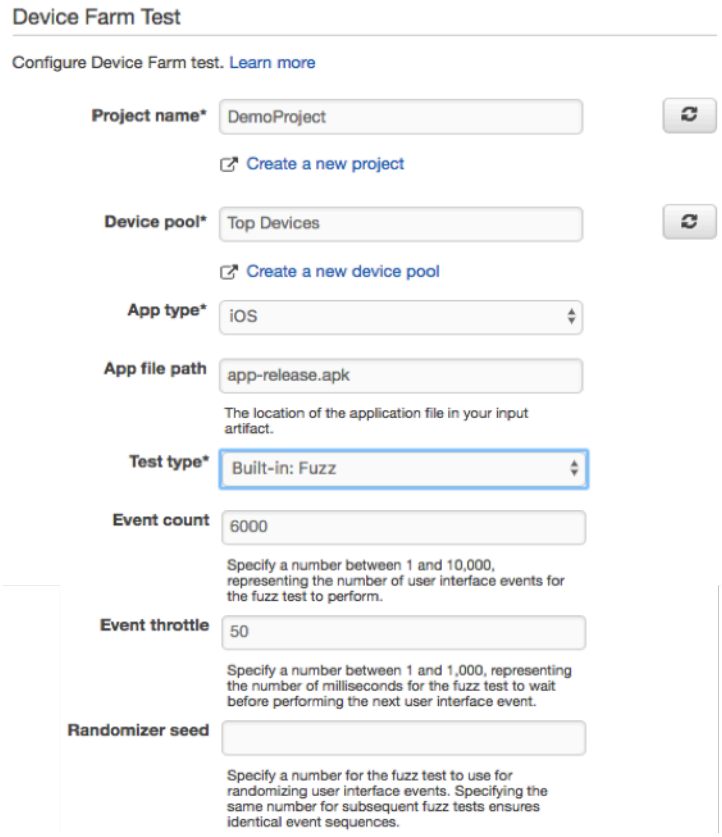
Para configurar um teste do Device Farm, execute como uma ação de teste do CodePipeline

1. Faça login no AWS Management Console e abra o console do CodePipeline 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.



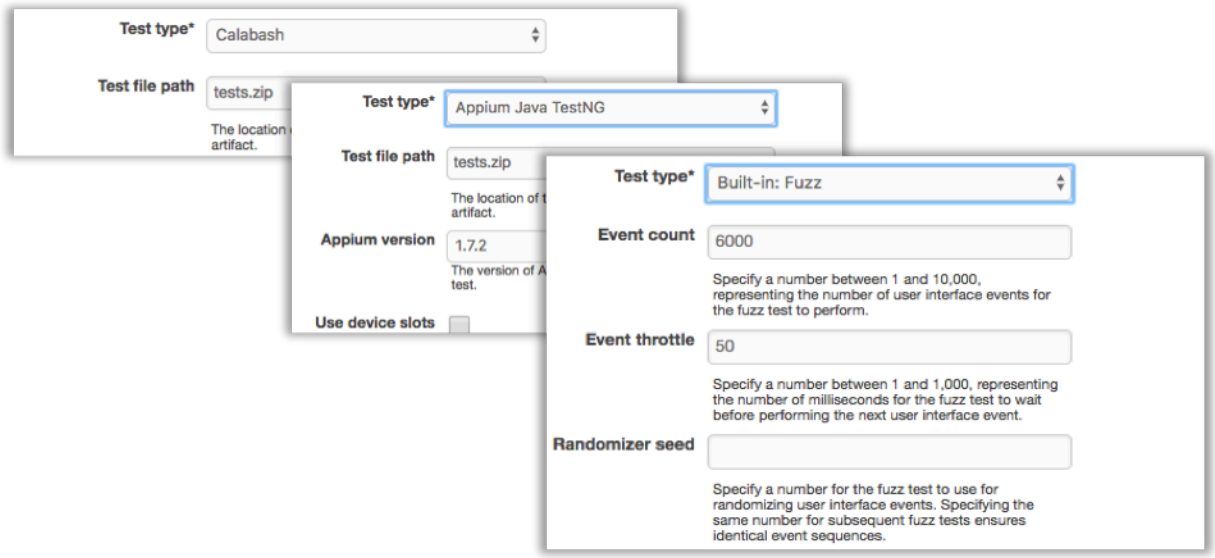
7. Em Nome do projeto, selecione o projeto existente do ou selecione Criar um novo projeto.
8. Em Grupo de dispositivos, selecione seu grupo de dispositivos existente ou selecione Criar um novo grupo de dispositivos. Se você criar um grupo de dispositivos, será necessário selecionar um conjunto de dispositivos de teste.

9. Em Tipo de aplicativo, selecione a plataforma para seu aplicativo.

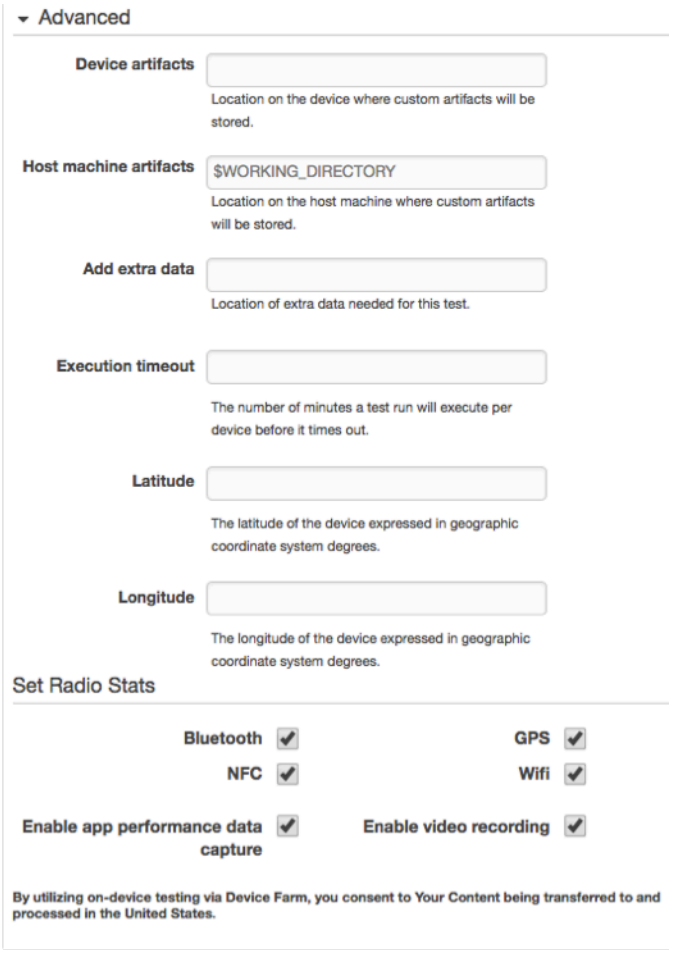


10. Em Caminho para o arquivo do aplicativo, insira o caminho do pacote do aplicativo compilado. O caminho é relativo à raiz do artefato de entrada para o teste.
11. Em Tipo de teste, siga um destes procedimentos:

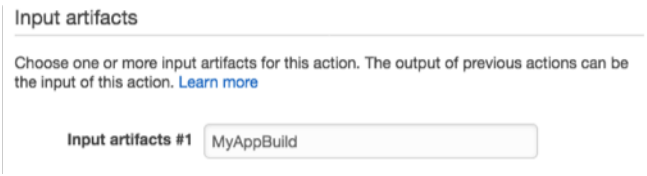
    - Se estiver usando um dos testes internos do Device Farm, escolha o tipo de teste configurado no projeto do Device Farm.
    - Se não estiver usando um dos testes incorporados do Device Farm, no caminho do arquivo de teste, insira o caminho do arquivo de definição do teste. O caminho é relativo à raiz do artefato de entrada para o teste.



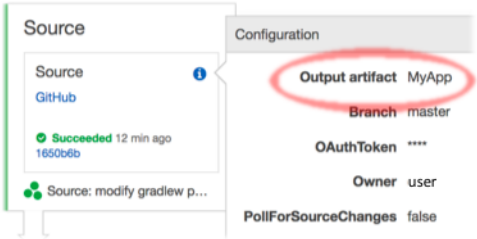
12. Nos campos restantes, forneça a configuração que seja apropriada para seu teste e tipo de aplicativo.
13. (Opcional) Em Avançado, forneça uma configuração detalhada para a execução do teste.



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 console do CodePipeline, 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 o pipeline testa o aplicativo diretamente no estágio Origem, selecione MyApp. Se o pipeline inclui um estágio de Compilação, selecione MyAppBuild.



15. Na parte inferior do painel, selecione Adicionar ação.
16. No painel CodePipeline, selecione Salvar alteração do pipeline e, em seguida, selecione Salvar alteração.
17. Para enviar suas alterações e iniciar uma compilação do pipeline, selecione Liberar alteração e, depois, Liberar.

Referência do AWS CLI para o AWS Device Farm

Para usar a AWS Command Line Interface (AWS CLI) para executar comandos do Device Farm, consulte a Referência da AWS CLI para o AWS Device Farm.

Para obter informações gerais sobre a AWS CLI, consulte o Guia do usuário do AWS Command Line Interface e a Referência de comandos do AWS CLI.

Referência do Windows PowerShell para o AWS Device Farm

Para usar o Windows PowerShell para executar comandos do Device Farm, consulte a Referência de Cmdlet do Device Farm na Referência de Cmdlet do AWS Tools for Windows PowerShell. Para obter mais informações, consulte Configuração das ferramentas do AWS para Windows PowerShell no Guia do Usuário do AWS Tools for Windows PowerShell.

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 a AWS CLI fornecem meios para fazer isso.

O AWS SDK fornece acesso a todos os serviços da AWS, incluindo Device Farm, Amazon S3 e muito mais. Para obter mais informações, consulte

- as ferramentas e os SDKs da AWS
- Referência da API do AWS Device Farm

Exemplo: Uso da AWS SDK para iniciar uma execução do Device Farm e coletar artefatos

O exemplo a seguir fornece uma demonstração do início ao fim de como você pode usar o AWS SDK para trabalhar com o Device Farm. Esse exemplo faz o seguinte:

- Carrega um teste e pacotes de aplicativos 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 aplicativos 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 Trabalho com testes do 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 Trabalho com testes do 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 a partir 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 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 Trabalho com testes do 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 a partir 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 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 Trabalho com testes do 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.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 Trabalho com testes do 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.xmle` 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 Trabalho com testes do 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 `FLAG_SECURE` recurso do Android. Esse sinalizador (conforme descrito na documentação oficial do Android) é usado para impedir que determinadas janelas de um aplicativo 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 do seu aplicativo se a janela usar esse sinalizador.

Esse sinalizador é frequentemente usado por desenvolvedores para páginas em seus aplicativos que contêm informações confidenciais, como páginas de login. Se você estiver vendo uma tela preta no lugar da tela do seu aplicativo para determinadas páginas, como a página de login, trabalhe com seus desenvolvedores para obter uma versão do aplicativo que não use esse sinalizador para testes.

Além disso, observe que o Device Farm ainda pode interagir com janelas de aplicativos que tenham esse sinalizador. Portanto, se a página de login do seu aplicativo aparecer como uma tela preta, você ainda poderá inserir suas credenciais para fazer login no aplicativo (e, assim, visualizar páginas não bloqueadas pela `FLAG_SECURE` bandeira).

Solução de problemas de testes Appium Java JUnit no AWS Device Farm

O tópico a seguir lista mensagens de erro que ocorrem durante o upload de testes do Appium Java JUnit e recomenda soluções para resolver cada erro.

> **Note**
>
> As instruções a seguir baseiam-se no Linux x86_64 e Mac.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_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 pacote válido do Appium Java JUnit deve gerar um resultado 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)
      |— 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 Trabalhando com a Appium e o 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 pacote do Appium Java JUnit for válido, você encontrará o diretório *dependency-jars* no 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 Trabalhando com a Appium e o 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 pacote do Appium Java JUnit for válido, você encontrará pelo menos um arquivo *jar* no diretório *dependency-jars*:

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

Para obter mais informações, consulte Trabalhando com a Appium e o 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 pacote do Appium Java JUnit for válido, você encontrará pelo menos um arquivo *jar* semelhante a *acme-android-appium-1.0-SNAPSHOT-tests.jar* em 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 Trabalhando com a Appium e o 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 semelhante a *acme-android-appium-1.0-SNAPSHOT-tests.jar* em 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 Trabalhando com a Appium e o 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 o valor da versão do JUnit. Descompacte o pacote de testes e abra o diretório dependency-jars, verifique se o arquivo JAR do JUnit 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 .
   ```

   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 pacote do Appium Java JUnit for válido, você encontrará o arquivo de dependência JUnit, que é semelhante ao arquivo jar *junit-4.10.jar* em nosso exemplo. O nome deve conter a palavra-chave *junit* e o número da versão, que no exemplo é 4.10.

Para obter mais informações, consulte Trabalhando com a Appium e o 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**
>
> Percebemos que a versão do JUnit era anterior à versão mínima compatível 4.10. Altere a versão do JUnit 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 dependência JUnit semelhante a *junit-4.10.jar* em nosso exemplo, bem como o número da versão, que no 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**
>
> Talvez os testes não executem corretamente se a versão do JUnit especificada em seu pacote de testes for anterior à versão mínima compatível 4.10.

Para obter mais informações, consulte Trabalhando com a Appium e o AWS Device Farm.

Solução de problemas de testes de aplicativos da Web Appium Java JUnit no AWS Device Farm

O tópico a seguir lista mensagens de erro que ocorrem durante o upload dos testes do Appium Java JUnit para aplicativos web e recomenda soluções 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 pacote válido do Appium Java JUnit deve gerar um resultado 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)
      |— 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 pacote do Appium Java JUnit for válido, você encontrará o diretório *dependency-jars* no 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 pacote do Appium Java JUnit for válido, você encontrará pelo menos um arquivo *jar* no diretório *dependency-jars*:

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

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

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 pacote do Appium Java JUnit for válido, você encontrará pelo menos um arquivo *jar* semelhante a *acme-android-appium-1.0-SNAPSHOT-tests.jar* em 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 semelhante a *acme-android-appium-1.0-SNAPSHOT-tests.jar* em 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 o valor da versão do JUnit. Descompacte o pacote de testes e abra o diretório dependency-jars, verifique se o arquivo JAR do JUnit 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 .
   ```

   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 pacote do Appium Java JUnit for válido, você encontrará o arquivo de dependência JUnit, que é semelhante ao arquivo jar *junit-4.10.jar* em nosso exemplo. O nome deve conter a palavra-chave *junit* e o número da versão, que no 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**
>
> Percebemos que a versão do JUnit era anterior à versão mínima compatível 4.10. Altere a versão do JUnit 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 dependência JUnit semelhante a *junit-4.10.jar* em nosso exemplo, bem como o número da versão, que no 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**
>
> Talvez os testes não executem corretamente se a versão do JUnit especificada em seu pacote de testes for anterior à versão mínima compatível 4.10.

Para obter mais informações, consulte Trabalhando com a Appium e o 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 pacote válido do Appium Java JUnit deve gerar um resultado 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)
      |— 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 Trabalhando com a Appium e o 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 pacote do Appium Java JUnit for válido, você encontrará o diretório *dependency-jars* no 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 Trabalhando com a Appium e o 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 pacote do Appium Java JUnit for válido, você encontrará pelo menos um arquivo *jar* no diretório *dependency-jars*.

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

Para obter mais informações, consulte Trabalhando com a Appium e o 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 pacote do Appium Java JUnit for válido, você encontrará pelo menos um arquivo *jar* semelhante a *acme-android-appium-1.0-SNAPSHOT-tests.jar* em 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 Trabalhando com a Appium e o 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 semelhante a *acme-android-appium-1.0-SNAPSHOT-tests.jar* em 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 Trabalhando com a Appium e o AWS Device Farm.

Solução de problemas de aplicativos da 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 pacote válido do Appium Java JUnit deve gerar um resultado 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)
      |— 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 Trabalhando com a Appium e o 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 pacote do Appium Java JUnit for válido, você encontrará o diretório *dependency-jars* no 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 Trabalhando com a Appium e o 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 pacote do Appium Java JUnit for válido, você encontrará pelo menos um arquivo *jar* no diretório *dependency-jars*.

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

Para obter mais informações, consulte Trabalhando com a Appium e o 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 pacote do Appium Java JUnit for válido, você encontrará pelo menos um arquivo *jar* semelhante a *acme-android-appium-1.0-SNAPSHOT-tests.jar* em 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 Trabalhando com a Appium e o 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 semelhante a *acme-android-appium-1.0-SNAPSHOT-tests.jar* em 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 Trabalhando com a Appium e o 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 Trabalhando com a Appium e o 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 do Appium Python for válido, você encontrará pelo menos um arquivo dependente *.whl*, como os arquivos destacados no 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 Trabalhando com a Appium e o 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 do Appium Python for válido, você encontrará pelo menos um arquivo dependente *.whl*, como os arquivos destacados no 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 Trabalhando com a Appium e o 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 do Appium Python for válido, você encontrará o diretório *tests* no 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 Trabalhando com a Appium e o 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 do Appium Python for válido, você encontrará o diretório *tests* no diretório de trabalho. O nome do arquivo pode ser diferente, mas deve terminar com *test_* ou 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 Trabalhando com a Appium e o 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 do Appium Python for válido, você encontrará o arquivo *requirement.txt* no 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 Trabalhando com a Appium e o 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 arquivo *requirements.txt* no 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 Trabalhando com a Appium e o 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 Trabalhando com a Appium e o 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 Trabalhando com a Appium e o AWS Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEELS_INSUFICIENTE

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar dependências de roda suficientes no diretório wheelhouse. Descompacte seu pacote de teste e, em seguida, abra o diretório da casa do leme. Verifique se você tem todas as dependências de roda 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 arquivo *requirements.txt*, bem como o número de arquivos dependentes *de.whl* no diretório wheelhouse:

```
$ cat requirements.txt | egrep "." |wc -l
    12
$ ls wheelhouse/ | egrep ".+\.whl" | wc -l
    11
```

Se o número de arquivos dependentes *de.whl* for menor que o número de linhas não vazias em seu arquivo *requirements.txt*, você precisará garantir o seguinte:

- Há um arquivo dependente *de.whl* correspondente a cada linha no arquivo *requirements.txt*.
- Não há outras linhas no arquivo *requirements.txt* que contenham informações além dos nomes dos pacotes de dependências.
- Nenhum nome de dependência é duplicado em várias linhas no arquivo *requirements.txt*, de forma que duas linhas no arquivo possam corresponder a um arquivo dependente *de.whl.*

O AWS Device Farm não suporta linhas no arquivo *requirements.txt* que não correspondam diretamente aos pacotes de dependência, como linhas que especificam opções globais para o `pip install` comando. Consulte Formato de arquivo de requisitos para obter uma lista de opções globais.

Para obter mais informações, consulte Trabalhando com a Appium e o AWS Device Farm.

Solução de problemas de testes de aplicativos 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 Trabalhando com a Appium e o 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 do Appium Python for válido, você encontrará pelo menos um arquivo dependente *`.whl`*, como os arquivos destacados no 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 Trabalhando com a Appium e o 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 do Appium Python for válido, você encontrará pelo menos um arquivo dependente *`.whl`*, como os arquivos destacados no 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 Trabalhando com a Appium e o 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 do Appium Python for válido, você encontrará o diretório *tests* no 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 Trabalhando com a Appium e o 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 do Appium Python for válido, você encontrará o diretório *tests* no diretório de trabalho. O nome do arquivo pode ser diferente, mas deve terminar com *test_* ou 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 Trabalhando com a Appium e o 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 do Appium Python for válido, você encontrará o arquivo *requirement.txt* no 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 Trabalhando com a Appium e o 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 arquivo *requirement.txt* no 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 Trabalhando com a Appium e o 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 Trabalhando com a Appium e o 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 Trabalhando com a Appium e o 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.

INSTRUMENTATION_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 APK. 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 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 Trabalhar com 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.

> **Warning**
>
> Não foi possível extrair informações sobre o pacote de teste. Para verificar se o pacote de testes é válido, execute o comando "aapt debug badging <caminho para seu pacote de testes>" e tente novamente se o comando não imprimir nenhum erro.

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 Trabalhar com 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.

> **Warning**
>
> Não foi possível localizar o valor do executor de instrumentação no AndroidManifest.xml. Para verificar se o pacote de testes é válido, execute o comando "aapt dump xmltree <caminho para seu pacote de testes> AndroidManifest.xml" e tente novamente depois que encontrar o valor do executor de instrumentação subjacente à palavra-chave "instrumentation".

Durante o processo de validação de upload, o Device Farm analisa o valor do corredor 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 Trabalhar com 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.

> **Warning**
>
> Não foi possível encontrar o AndroidManifest.xml válido em seu pacote de testes. Para verificar se o pacote de testes é válido, execute o comando "aapt dump xmltree <caminho para seu pacote de testes> AndroidManifest.xml" e tente novamente se o comando não imprimir nenhum erro.

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 Trabalhar com 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.

> **Warning**
>
> Não foi possível encontrar o nome do pacote em seu pacote de testes. Para verificar se o pacote de testes é válido, execute o comando "aapt debug badging <caminho para seu pacote de testes>" e tente novamente depois que encontrar o valor do nome do pacote subjacente à palavra-chave "package: name".

Durante o processo de validação do upload, o Device Farm analisa o valor do nome do pacote a partir da 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 Trabalhar com instrumentação para Android e AWS Device Farm.

Solução de problemas de testes de aplicativos 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 AWSDeviceFarmiOSReferenceApp.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 Trabalho com 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 AWSDeviceFarmiOSReferenceApp.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 de aplicativos iOS for válido, você encontrará o diretório *Payload* no diretório de trabalho.

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

   Para obter mais informações, consulte Trabalho com 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 AWSDeviceFarmiOSReferenceApp.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 de aplicativos iOS for válido, você encontrará um diretório *.app* semelhante a *AWSDeviceFarmiOSReferenceApp.app* em nosso exemplo no diretório *Payload*.

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

Para obter mais informações, consulte Trabalho com 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 AWSDeviceFarmiOSReferenceApp.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 de aplicativos iOS for válido, você encontrará o arquivo *Info.plist* no diretório *.app*, semelhante a *AWSDeviceFarmiOSReferenceApp.app* em nosso exemplo.

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

Para obter mais informações, consulte Trabalho com 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 o pacote de aplicativos e abra o arquivo Info.plist no diretório .app, verifique se a chave "UIRequiredDeviceCapabilities" está especificada e tente novamente.

No exemplo a seguir, o nome do pacote é AWSDeviceFarmiOSReferenceApp.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 arquivo *Info.plist* em um diretório *.app* semelhante a *AWSDeviceFarmiOSReferenceApp.app* em 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 Trabalho com 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 o pacote de aplicativos e abra o arquivo Info.plist no diretório .app, verifique se a chave "CFBundleSupportedPlatforms" está especificada e tente novamente.

No exemplo a seguir, o nome do pacote é AWSDeviceFarmiOSReferenceApp.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 arquivo *Info.plist* em um diretório *.app* semelhante a *AWSDeviceFarmiOSReferenceApp.app* em 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 Trabalho com 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 o pacote de aplicativos e abra o arquivo Info.plist no diretório .app, verifique se o valor da chave "CFBundleSupportedPlatforms" não contém a palavra-chave "simulator" e tente novamente.

No exemplo a seguir, o nome do pacote é AWSDeviceFarmiOSReferenceApp.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 arquivo *Info.plist* em um diretório *.app* semelhante a *AWSDeviceFarmiOSReferenceApp.app* em 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 Trabalho com 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 o pacote de aplicativos e abra o arquivo Info.plist no diretório .app, verifique se a chave "UIDeviceFamily" está especificada e tente novamente.

No exemplo a seguir, o nome do pacote é AWSDeviceFarmiOSReferenceApp.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 arquivo *`Info.plist`* em um diretório *`.app`* semelhante a *`AWSDeviceFarmiOSReferenceApp.app`* em 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 Trabalho com 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 o pacote de aplicativos e abra o arquivo Info.plist no diretório .app, verifique se a chave "CFBundleIdentifier" está especificada e tente novamente.

No exemplo a seguir, o nome do pacote é AWSDeviceFarmiOSReferenceApp.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 arquivo *Info.plist* em um diretório *.app* semelhante a *AWSDeviceFarmiOSReferenceApp.app* em 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 Trabalho com 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 o pacote de aplicativos e abra o arquivo Info.plist no diretório .app, verifique se a chave "CFBundleExecutable" está especificada e tente novamente.

No exemplo a seguir, o nome do pacote é AWSDeviceFarmiOSReferenceApp.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 arquivo *Info.plist* em um diretório *.app* semelhante a *AWSDeviceFarmiOSReferenceApp.app* em 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 Trabalho com testes de iOS no AWS Device Farm.

Solução de problemas de testes do XCTest no AWS Device Farm

O tópico a seguir lista mensagens de erro que ocorrem durante o upload de testes do XCTest e recomenda soluções 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 pacote válido do XCTest deve gerar um resultado semelhante ao seguinte:

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

Para obter mais informações, consulte Trabalhando com o XCTest para iOS e AWS Device Farm.

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 pacote do XCTest for válido, você encontrará um diretório com um nome semelhante a *`swiftExampleTests.xctest`* no diretório de trabalho. O nome deve terminar com *`.xctest`*.

   ```
   .
   ```

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

Para obter mais informações, consulte Trabalhando com o XCTest para iOS e AWS Device Farm.

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 pacote do XCTest for válido, você encontrará o arquivo *Info.plist* no diretório *.xctest*. Em nosso exemplo a seguir, o diretório é chamado de *swiftExampleTests.xctest*.

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

Para obter mais informações, consulte Trabalhando com o XCTest para iOS e AWS Device Farm.

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 o pacote de testes 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 arquivo *Info.plist* em um diretório *.xctest* semelhante a *swiftExampleTests.xctest* em 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 aplicativos válido do XCTest deve gerar um resultado semelhante ao seguinte:

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

Para obter mais informações, consulte Trabalhando com o XCTest para iOS e AWS Device Farm.

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 testes e abra o arquivo Info.plist, verifique se a chave "CFBundleExecutable" 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 arquivo *Info.plist* em um diretório *.xctest* semelhante a *swiftExampleTests.xctest* em 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 aplicativos válido do XCTest deve gerar um resultado semelhante ao seguinte:

   ```
   swiftExampleTests
   ```

   Para obter mais informações, consulte Trabalhando com o XCTest para iOS e AWS Device Farm.

Solução de problemas de testes de interface do usuário do XCTest no AWS Device Farm

O tópico a seguir lista mensagens de erro que ocorrem durante o upload de testes do XCTest UI e recomenda soluções 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.

> **Warning**
>
> Não conseguimos abrir seu arquivo de teste IPA. 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 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 XCTest UI.

XCTEST_UI_TEST_PACKAGE_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 testes. Descompacte o pacote de testes, verifique se o diretório Payload encontra-se no pacote e tente novamente.

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 do XCTest UI for válido, você encontrará o diretório *`Payload`* no 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 XCTest UI.

XCTEST_UI_TEST_PACKAGE_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 testes 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 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 do XCTest UI for válido, você encontrará um diretório *.app* semelhante a *swift-sampleUITests-Runner.app* em nosso exemplo no diretório *Payload*.

   ```
   .
   `-- 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 XCTest UI.

XCTEST_UI_TEST_PACKAGE_PLUGINS_DIR_MISSING

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar o diretório Plugins no diretório .app. Descompacte o pacote de testes e abra o diretório .app, verifique se o diretório Plugins encontra-se no diretório e tente novamente.

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 do XCTest UI for válido, você encontrará o diretório *Plugins* em um diretório *.app*. 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 XCTest UI.

XCTEST_UI_TEST_PACKAGE_XCTEST_DIR_MISSING_IN_PLUGINS_DIR

Se você visualizar a mensagem a seguir, siga estas etapas para corrigir o problema.

> **Warning**
>
> Não foi possível encontrar o diretório .xctest no diretório Plugins. Descompacte o pacote de testes e abra o diretório Plugins, verifique se o diretório .xctest encontra-se no diretório e tente novamente.

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 do XCTest UI for válido, você encontrará um diretório *`.xctest`* no diretório *`Plugins`*. 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 XCTest UI.

XCTEST_UI_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 .app. Descompacte o pacote de testes 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 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 do XCTest UI for válido, você encontrará o arquivo *Info.plist* no diretório *.app*. Em nosso exemplo a seguir, 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 XCTest UI.

XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING_IN_XCTEST_DIR

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 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 do XCTest UI for válido, você encontrará o arquivo *Info.plist* no diretório *.xctest*. Em nosso exemplo a seguir, 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 XCTest UI.

XCTEST_UI_TEST_PACKAGE_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 o pacote de testes e abra o arquivo Info.plist no diretório .app, verifique se a chave "UIRequiredDeviceCapabilities" está especificada e tente novamente.

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 arquivo *Info.plist* em um diretório *.app* semelhante a *swift-sampleUITests-Runner.app* em 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 válido do XCTest UI deve gerar um resultado semelhante ao seguinte:

```
['armv7']
```

Para obter mais informações, consulte XCTest UI.

XCTEST_UI_TEST_PACKAGE_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 Info.plist. Descompacte o pacote de testes e abra o arquivo Info.plist no diretório .app, verifique se a chave "CFBundleSupportedPlatforms" está especificada e tente novamente.

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 arquivo *Info.plist* em um diretório *.app* semelhante a *swift-sampleUITests-Runner.app* em 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 válido do XCTest UI deve gerar um resultado semelhante ao seguinte:

   ```
   ['iPhoneOS']
   ```

   Para obter mais informações, consulte XCTest UI.

XCTEST_UI_TEST_PACKAGE_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 o pacote de testes e abra o arquivo Info.plist no diretório .app, verifique se o valor da chave "CFBundleSupportedPlatforms" não contém a palavra-chave "simulator" e tente novamente.

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 arquivo *Info.plist* em um diretório *.app* semelhante a *swift-sampleUITests-Runner.app* em 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 válido do XCTest UI deve gerar um resultado semelhante ao seguinte:

```
['iPhoneOS']
```

Se o pacote do XCTest UI for válido, o valor não deve conter a palavra-chave `simulator`.

Para obter mais informações, consulte XCTest UI.

XCTEST_UI_TEST_PACKAGE_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 de formato no Info.plist. Descompacte o pacote de testes e abra o arquivo Info.plist no diretório .app, verifique se a chave "UIDeviceFamily" está especificada e tente novamente.

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 arquivo *Info.plist* em um diretório *.app* semelhante a *swift-sampleUITests-Runner.app* em 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 válido do XCTest UI deve gerar um resultado semelhante ao seguinte:

```
[1, 2]
```

Para obter mais informações, consulte XCTest UI.

XCTEST_UI_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 o pacote de testes e abra o arquivo Info.plist no diretório .app, verifique se a chave "CFBundleIdentifier" está especificada e tente novamente.

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 arquivo *Info.plist* em um diretório *.app* semelhante a *swift-sampleUITests-Runner.app* em 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 válido do XCTest UI deve gerar um resultado semelhante ao seguinte:

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

   Para obter mais informações, consulte XCTest UI.

XCTEST_UI_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 testes e abra o arquivo Info.plist no diretório .app, verifique se a chave "CFBundleExecutable" está especificada e tente novamente.

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 arquivo *Info.plist* em um diretório *.app* semelhante a *swift-sampleUITests-Runner.app* em 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 válido do XCTest UI deve gerar um resultado semelhante ao seguinte:

```
XCTRunner
```

Para obter mais informações, consulte XCTest UI.

XCTEST_UI_TEST_PACKAGE_TEST_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 no diretório .xctest. Descompacte o pacote de testes e abra o arquivo Info.plist no diretório .xctest, verifique se a chave "CFBundleIdentifier" está especificada e tente novamente.

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 arquivo *Info.plist* em um diretório *.app* semelhante a *swift-sampleUITests-Runner.app* em 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 válido do XCTest UI deve gerar um resultado semelhante ao seguinte:

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

   Para obter mais informações, consulte XCTest UI.

XCTEST_UI_TEST_PACKAGE_TEST_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 no diretório .xctest. Descompacte o pacote de testes e abra o arquivo Info.plist no diretório .xctest, verifique se a chave "CFBundleExecutable" está especificada e tente novamente.

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 arquivo *Info.plist* em um diretório *.app* semelhante a *swift-sampleUITests-Runner.app* em 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 válido do XCTest UI deve gerar um resultado semelhante ao seguinte:

```
swift-sampleUITests
```

Para obter mais informações, consulte XCTest UI.

Segurança em AWS Device Farm

A segurança para com a nuvem na AWS é a nossa maior prioridade. Como cliente da AWS, você contará com um datacenter e uma arquitetura de rede criados para atender aos requisitos das organizações com as maiores exigências de segurança.

A segurança é uma responsabilidade compartilhada entre a AWS e você. O modelo de responsabilidade compartilhada descreve isso como segurança da nuvem e segurança na nuvem:

- Segurança da nuvem: a AWS é responsável pela proteção da infraestrutura que executa produtos da AWS na Nuvem AWS. A AWS também fornece serviços que podem ser usados com segurança. Auditores de terceiros testam e verificam regularmente a eficácia da nossa segurança como parte dos Programas de conformidade da AWS. Para saber mais sobre os programas de conformidade que se aplicam ao AWS Device Farm, consulte Serviços da AWS no escopo por 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 sua 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

Público

O modo como você usa o 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. Entender como o acesso é gerenciado pode ajudá-lo a solicitar as permissões corretas ao seu 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. Assim, você deve enviar solicitações ao administrador do IAM para alterar as permissões dos usuários de seu serviço. Revise as informações nesta página para entender os Introdução ao 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 de IAM - Se você for um administrador de 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.

Autenticando com identidades

A autenticação é a forma como você faz login na AWS usando suas credenciais de identidade. É necessário ser autenticado (fazer login na AWS) como o usuário raiz da Usuário raiz da conta da AWS, como usuário do IAM ou assumindo um perfil do IAM.

Você pode fazer login na AWS como uma identidade federada usando credenciais fornecidas por uma fonte de identidades. AWS IAM Identity Center Os usuários do IAM Identity Center, a autenticação única da empresa e as suas credenciais do Google ou do Facebook são exemplos de identidades federadas. Quando você faz login como uma identidade federada, o administrador já configurou anteriormente a federação de identidades usando perfis do IAM. Quando você acessa a AWS usando a federação, está indiretamente assumindo um perfil.

É possível fazer login no AWS Management Console ou no portal de acesso da AWS dependendo do tipo de usuário que você é. Para obter mais informações sobre como fazer login na AWS, consulte Como fazer login na conta da Conta da AWS no Guia do usuário do Início de Sessão da AWS.

Se você acessar a AWS programaticamente, a AWS fornecerá um kit de desenvolvimento de software (SDK) e uma interface da linha de comando (CLI) para você assinar criptograficamente as solicitações usando as suas credenciais. Se você não utilizar as ferramentas da AWS, deverá assinar as solicitações por conta própria. Para obter mais informações sobre como usar o método recomendado para assinar solicitações por conta própria, consulte Assinar solicitações de API da AWS no Guia do usuário do IAM.

Independentemente do método de autenticação usado, também pode ser exigido que você forneça mais informações de segurança. Por exemplo, a AWS recomenda o uso da autenticação multifator (MFA) para aumentar a segurança de sua conta. Para saber mais, consulte Autenticação multifator no GuiaAWS IAM Identity Center do usuário. Usar a autenticação multifator (MFA) naAWS no Guia do usuário do IAM.

Usuário root da Conta da AWS

Ao criar uma Conta da AWS, você começa com uma identidade de login que tem acesso completo a todos os recursos e Serviços da AWS na conta. Essa identidade, denominada usuário raiz da Conta da AWS, e é acessada por 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 pode 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 Conta da AWS que tem permissões específicas para uma única pessoa ou aplicação. Sempre que possível, recomendamos depender de 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, recomendamos alternar as chaves de acesso. Para obter mais informações, consulte Alterne 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 IAMAdmins e atribuir a esse grupo permissões para administrar recursos do IAM.

Usuários são diferentes de funções. Um usuário é exclusivamente associado a uma pessoa ou a uma aplicação, mas uma função pode ser assumida por qualquer pessoa que precisar dela. Os usuários têm credenciais permanentes de longo prazo, mas as funções fornecem credenciais temporárias. Para saber mais, consulte Quando criar um usuário do IAM (em vez de uma função) no Guia do usuário do IAM.

Funções do IAM

Um perfil do IAM é uma identidade dentro da Conta da AWS que tem permissões específicas. Ela é semelhante a um usuário do IAM, mas não está associada a uma pessoa específica. É possível assumir temporariamente um perfil do IAM no AWS Management Console alternando perfis. É possível assumir um perfil chamando uma operação de API da AWS CLI ou da AWS, ou usando um URL personalizado. Para obter mais informações sobre métodos para o uso de perfis, consulte Usar perfis do IAM no Guia do usuário do IAM.

Os perfis do IAM com credenciais temporárias são úteis nas seguintes situações:

- Acesso de usuário federado: para atribuir permissões a identidades federadas, você pode 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 pelo mesmo. Para obter mais informações sobre perfis para federação, consulte Criar um perfil para um provedor de identidades de terceiros no Guia do usuário do IAM. Se você usar o IAM Identity Center, configure um conjunto de permissões. Para controlar o que suas identidades podem acessar após a autenticação, o IAM Identity Center 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 assumir 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. As funções são a principal forma de conceder acesso entre contas. No entanto, alguns Serviços da AWS permitem que você anexe uma política diretamente a um recurso (em vez de usar uma função como proxy).

Para saber a diferença entre funções e políticas baseadas em recurso para acesso entre contas, consulte Como as funções do IAM diferem das políticas baseadas em recurso 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 aplicações no Amazon EC2 ou armazene objetos no Amazon S3. Um serviço pode fazer isso usando as permissões do principal de chamada, usando um perfil de serviço ou um perfil vinculado ao serviço.
  - Encaminhamento de sessões de acesso (FAS): qualquer pessoa que utilizar uma função ou usuário do IAM para realizar ações na AWS é considerada uma entidade principal. Ao usar alguns serviços, você pode executar uma ação que inicia outra ação em um serviço diferente. O recurso FAS utiliza as permissões da entidade principal que chama um AWS service (Serviço da AWS), combinadas às permissões do AWS service (Serviço da AWS) solicitante, para realizar solicitações para serviços downstream. As solicitações de FAS só são feitas quando um serviço recebe uma solicitação que exige interações com outros Serviços da AWS ou com recursos para serem concluídas. Nesse caso, você precisa ter permissões para executar ambas as ações. Para obter detalhes da política ao fazer solicitações de FAS, consulte Encaminhar sessões de acesso.
  - Perfil de serviço: um perfil de serviço é um perfil do IAM que um serviço assume para realizar 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.
  - Perfil vinculado ao serviço: um perfil vinculado ao serviço é um tipo de perfil de serviço vinculado a um AWS service (Serviço da AWS). O serviço pode assumir o perfil para executar uma ação em seu nome. Os perfis vinculados ao serviço aparecem em sua Conta da AWS e são de propriedade do serviço. Um administrador do IAM pode visualizar, mas não pode editar as permissões para perfis vinculados a serviço.
- Aplicações em execução no Amazon EC2: é possível usar um perfil do IAM para gerenciar credenciais temporárias para aplicações em execução em uma instância do EC2 e fazer solicitações da AWS CLI ou da AWS API. É preferível fazer isso a armazenar chaves de acesso na instância do EC2. Para atribuir uma função da AWS a uma instância do EC2 e disponibilizá-la para todas as suas aplicações, crie um perfil de instância que esteja anexado a ela. Um perfil de instância contém a função e permite que os programas em execução na instância do EC2 obtenham credenciais temporárias. Para mais informações, consulte Usar um perfil do IAM para

conceder permissões a aplicações em execução nas instâncias do Amazon EC2 no Guia do usuário do IAM.

Para saber se deseja usar os perfis do IAM, consulte Quando criar um perfil do IAM (em vez de um usuário) 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 obter uma visão de alto nível de como o Device Farm e outros serviços AWS que funcionam com o IAM, consulte <> Serviços AWS que funcionam com o 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
- Funções do Device Farm IAM

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 as políticas JSON da AWS para especificar quem tem acesso a o 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 você pode usar para permitir ou negar acesso em uma política. As ações de política geralmente têm o mesmo nome que a operação de API da AWS associada. Existem algumas exceções, como ações somente de permissão, que não

têm uma operação de API correspondente. Há também algumas operações que 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 Selenium com a operação de API de teste de navegador de desktop do Device Farm `CreateTestGridUrl`, 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 Ações definidas por AWS Device Farm na Referência de autorização de serviço do IAM.

Recursos

Os administradores podem usar as políticas JSON da AWS para especificar quem tem acesso a quê. Ou seja, qual entidade principal pode executar ações em quais recursos, e em que condições.

O elemento `Resource` de política JSON especifica o objeto ou os objetos aos quais a ação se aplica. As instruções devem incluir um elemento `Resource` ou um elemento `NotResource`. Como prática recomendada, especifique um recurso usando seu Nome do recurso da Amazon (ARN). Isso pode ser feito para ações que oferecem suporte a um tipo de recurso específico, conhecido como permissões em nível de recurso.

Para ações que não oferecem suporte a permissões em nível de recurso, como operações de listagem, use um caractere curinga (*) para indicar que a instrução se aplica a todos os recursos.

```
"Resource": "*"
```

O recurso de instância do Amazon EC2 tem o seguinte ARN:

```
arn:${Partition}:ec2:${Region}:${Account}:instance/${InstanceId}
```

Para obter mais informações sobre o formato de ARNs, consulte Nomes do recurso da Amazon (ARNs) e namespaces de serviços da AWS.

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 da API do Amazon EC2 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 por 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 Ações definidas por AWS Device Farm na Referência de autorização de serviço do IAM.

Chaves de condição

Os administradores podem usar as políticas JSON AWS para especificar quem tem acesso a quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

O elemento `Condition` (ou bloco de `Condition`) permite que você especifique condições nas quais uma instrução está em vigor. O elemento `Condition` é opcional. É possível criar expressões condicionais que usam 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 `Condition` em uma instrução ou várias chaves em um único elemento `Condition`, a AWS os avaliará usando uma operação lógica `AND`. Se você especificar vários valores para uma única chave de condição, a AWS avaliará a condição usando uma operação lógica `OR`. Todas as condições devem ser atendidas para que as permissões da instrução sejam concedidas.

Você também pode usar variáveis de espaço reservado ao especificar as 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 de política do IAM: variáveis e tags no Guia do usuário do IAM.

A AWS oferece suporte a chaves de condição globais e chaves de condição específicas do serviço. Para ver todas as AWS chaves de condição globais da , consulte AWSChaves de contexto de condição globais da no Guia do usuário do IAM.

O Device Farm define seu próprio conjunto de chaves de condição e também suporta o uso de algumas chaves de condição globais. Para ver todas as chaves de condição globais da AWS, consulte Chaves de contexto de condição globais da AWS no Guia do usuário do IAM.

Para ver uma lista das chaves de condição do Device Farm, consulte Chaves de condição para 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 Ações definidas por 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 é compatível com 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 `aws:TagKeys` chaves de condição. 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 atributo baseado em tags desse atributo, consulte Visualizando projetos de teste do navegador de desktop Device Farm com base em tags.

Funções do Device Farm IAM

Uma função IAM é uma entidade dentro de sua conta da AWS 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 uma função IAM ou uma função entre contas. Obtenha credenciais de segurança temporárias chamando operações de API AWS STS tais como AssumeRole ou GetFederationToken.

Perfis vinculados ao serviço

Funções vinculadas ao serviço permitem que os serviços da AWS acessem recursos em outros serviços para concluir uma ação em seu nome. Os perfis vinculados a serviço aparecem na 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 funções vinculadas a serviços no recurso de teste do navegador de desktop Device Farm. Para obter informações sobre essas funções, consulte Usando funções vinculadas a serviços nos testes do navegador de desktop Device Farm no guia do desenvolvedor.

Perfis de serviço

O Device Farm não é compatível com funções 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 na 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.

Gerenciamento do acesso usando políticas

Você controla o acesso na AWS criando políticas e anexando-as a identidades ou recursos da AWS. Uma política é um objeto na AWS que, quando associado a uma identidade ou recurso, define suas permissões. A AWS avalia essas políticas quando uma entidade 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 são armazenadas na 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 AWS as políticas JSON da para especificar quem tem acesso a quê. Ou seja, qual entidade principal pode executar ações em quais recursos e em que condições.

Por padrão, usuários e funções não têm permissões. Para conceder aos usuários permissão para executar ações nos recursos de que eles precisam, um administrador do IAM pode criar políticas do IAM. O administrador pode então adicionar as políticas do IAM a 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 perfis do AWS Management Console, da AWS CLI ou da API da AWS.

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 funções podem realizar, em quais recursos e em que condições. Para saber como criar uma política baseada em identidade, consulte Criar políticas do IAM no Guia do usuário do IAM.

As políticas baseadas em identidade podem ser categorizadas ainda mais como políticas em linha ou políticas gerenciadas. As políticas em linha são anexadas diretamente a um único usuário, grupo ou função. As políticas gerenciadas são políticas independentes que podem ser anexadas a vários usuários, grupos e funções na Conta da AWS. As políticas gerenciadas incluem políticas gerenciadas pela AWS 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 do Device Farm AWS.

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

A AWS aceita tipos de política 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 e dos seus limites de permissões. As políticas baseadas em recurso que especificam o usuário ou a função 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 (UO) no AWS Organizations. O AWS Organizations é um serviço para agrupar e gerenciar centralmente várias Contas da AWS

pertencentes à sua empresa. Se você habilitar todos os recursos em uma organização, poderá aplicar políticas de controle de serviço (SCPs) a qualquer uma ou a todas as contas. O SCP limita as permissões para entidades em contas-membro, incluindo cada Usuário raiz da conta da AWS. Para obter mais informações sobre o Organizations e SCPs, consulte Como os SCPs funcionam no Guia do usuário do AWS Organizations.

- Políticas de sessão: são políticas avançadas que você transmite como um parâmetro quando cria de forma programática uma sessão temporária para uma função ou um usuário federado. As permissões da sessão resultante são a interseção das políticas baseadas em identidade do usuário ou da função e das políticas de sessão. As permissões também podem ser provenientes de uma política baseada em recurso. 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 a AWS determina se deve permitir uma solicitação quando há vários tipos de política envolvidos, consulte Lógica da avaliaçãode políticas no Guia do usuário do IAM.

Exemplos de políticas baseadas em identidade do AWS Device Farm

Por padrão, os usuários e funções do IAM não têm permissão para criar ou modificar recursos do Device Farm. Eles também não podem executar tarefas usando o AWS Management Console, a AWS CLI ou uma API da AWS. 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 políticas

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 a Conta da AWS. Ao criar ou editar políticas baseadas em identidade, siga estas diretrizes e recomendações:

- Comece com as políticas gerenciadas pela AWS e avance para as permissões de privilégio mínimo: para começar a conceder permissões a seus usuários e workloads, use as políticas gerenciadas pela AWS que concedem permissões para muitos casos de uso comuns. Eles estão disponíveis na sua Conta da AWS. Recomendamos que você reduza ainda mais as permissões definindo políticas gerenciadas pelo cliente da AWS 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 a ações de serviço, se elas forem usadas por meio de um AWS service (Serviço da AWS) específico, como o AWS CloudFormation. Para obter mais informações, consulte Elementos de política JSON do IAM: Condition no Manual 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 100 verificações de política e recomendações acionáveis para ajudar você 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 houver um cenário que exija usuários do IAM ou um usuário raiz em sua Conta da AWS, ative a MFA para obter segurança adicional. Para exigir a 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 você pode criar uma política que permite 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 de forma programática usando a AWS CLI ou a AWS API.

```
{
    "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 IAM da sua conta AWS acesso a um dos seus projetos de teste de navegador Device Farm destktop, `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 receber um erro no AWS Management Console informando que você não está autorizado a executar uma ação, entre em contato com o administrador para obter assistência. 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 IAM, `mateojackson`, tenta usar o console para exibir 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 uma função para o Device Farm.

Alguns Serviços da AWS permitem que você passe um perfil existente para o serviço, em vez de criar um novo perfil de serviço ou perfil vinculado 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 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 a função 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 você precisar de ajuda, entre em contato com seu administrador da AWS. Seu administrador é a pessoa que forneceu a você 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 a sua 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 criar uma entidade IAM (usuário ou função) para a pessoa ou o aplicativo que precisa de acesso. Elas usarão as credenciais dessa entidade para acessar a AWS. Em seguida, você deve anexar uma política à entidade que concede a ela as permissões corretas no Device Farm.

Para começar a usar imediatamente, consulte Criar os primeiros usuário e grupo delegados pelo IAM no Guia do usuário do IAM.

Quero permitir que pessoas fora da minha conta AWS acessem meus recursos do Device Farm

É possível criar um perfil que os usuários de outras contas ou pessoas fora da sua organização possam utilizar para acessar seus recursos. Você pode especificar quem é confiável para assumir a função. 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 conceder acesso a seus atributos em todas as Contas da AWS pertencentes a você, consulte Fornecimento de acesso a um usuário do IAM em outra Conta da AWS pertencente a você no Guia de usuário do IAM.
- Para saber como conceder acesso a seus recursos para terceiros Contas da AWS, consulte Fornecimento de acesso a Contas da AWS pertencentes 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 saber a diferença entre usar perfis e políticas baseadas em recursos para acesso entre contas, consulte Como os perfis do IAM diferem de políticas baseadas em recursos no Guia do usuário do IAM.

Validação de conformidade do AWS Device Farm

Auditores de terceiros avaliam a segurança e a conformidade do AWS Device Farm como parte de vários programas de conformidade da AWS. Isso inclui SOC, PCI, FedRAMP, HIPAA e outros. O AWS Device Farm não está no escopo de nenhum programa de conformidade da AWS.

Para obter uma lista de serviços da AWS no escopo de programas de conformidade específicos, consulte Serviços da AWS no escopo pelo programa de conformidade. Para obter informações gerais, consulte Programas de conformidade da AWS.

É possível fazer download de 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 sensibilidade de seus dados, pelos objetivos de conformidade de sua empresa e pelas leis e regulamentações aplicáveis. A 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.
- Recursos de conformidade da AWS: essa coleção de manuais e guias pode ser aplicada a seu setor e local.
- Avaliação de recursos com regras no AWS ConfigGuia do desenvolvedor – AWS Configavalia a conformidade das configurações de seus recursos com práticas internas, diretrizes do setor e regulamentos.

- AWS Security Hub: esse serviço da AWS fornece uma visão abrangente do estado de sua segurança na AWS que ajuda você a conferir sua conformidade com padrões e práticas recomendadas de segurança do setor.

Proteção de dados no AWS Device Farm

O modelo de responsabilidade compartilhada AWS aplica-se à proteção de dados à AWS Device Farm (Device Farm). Conforme descrito nesse modelo, a AWS é responsável por proteger a infraestrutura global que executa toda a Nuvem AWS. Você é responsável por manter o controle sobre seu conteúdo hospedado nessa infraestrutura. Você também é responsável pelas tarefas de configuração e gerenciamento de segurança dos Serviços da AWS que usa. Para ter mais informações sobre a privacidade de dados, consulte as Perguntas frequentes sobre privacidade de dados. Para ter mais informações sobre a proteção de dados na Europa, consulte a postagem do blog AWS Shared Responsibility Model and GDPR no Blog de segurança da AWS.

Para fins de proteção de dados, recomendamos que você proteja as credenciais da Conta da AWS e configure as contas de usuário individuais com o AWS IAM Identity Center ou o 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 da AWS. Exigimos TLS 1.2 e recomendamos TLS 1.3.
- Configure o registro em log das atividades da API e do usuário com o AWS CloudTrail.
- Use as soluções de criptografia da AWS, juntamente com todos os controles de segurança padrão dos 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 140-2 ao acessar a AWS por meio de uma interface de linha de comando ou uma API, use um endpoint do FIPS. Para ter mais informações sobre endpoints do FIPS, consulte Federal Information Processing Standard (FIPS) 140-2.

É altamente recomendável que nunca sejam colocadas informações de identificação confidenciais, como endereços de email dos seus clientes, em marcações ou campos de formato livre, como um

campo Nome. Isso inclui quando você trabalha com o Device Farm ou outros Serviços da AWS usando o console, a API, AWS CLI ou AWS SDKs. Quaisquer dados inseridos em tags ou 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 pontos de extremidade do Device Farm suportam apenas solicitações HTTPS (SSL/TLS) assinadas, exceto quando indicado de outra forma. Todo o conteúdo recuperado ou colocado no Amazon S3 por meio de URLs de upload é criptografado usando SSL/TLS. Para obter mais informações sobre como as solicitações HTTPS são assinadas na AWS, consulte Assinar solicitações de API da AWS na Referência geral da AWS.

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 inativa

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. Após a expiração do período de retenção, os dados são removidos do armazenamento de backup do Device Farm, mas todos os metadados (ARNs, datas de upload, nomes de arquivos etc.) são preservados para uso futuro. A tabela a seguir lista o período de retenção para vários tipos de conteúdo.

Tipo de conteúdo	Período de retenção (dias)
Aplicativos carregados	30
Pacotes de teste carregados	30
Logs	400

Tipo de conteúdo	Período de retenção (dias)
Gravações de vídeo e outros artefatos	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 realizadas 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 registros 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 ter mais informações, consulte Trabalhando com 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 do Amazon VPC para se conectar aos seus recursos na AWS O acesso a qualquer infraestrutura não pública associada à sua conta (por exemplo, instâncias do Amazon EC2 sem um endereço IP público) deve usar um endpoint do Amazon VPC. Independentemente da configuração do endpoint VPC, o Device Farm isola seu tráfego de outros usuários em toda a rede do Device Farm.

Suas conexões fora da rede da AWS não são garantidas como protegidas ou seguras, e é sua responsabilidade proteger quaisquer conexões de Internet que seus aplicativos façam.

Resiliência no AWS Device Farm

A infraestrutura global da AWS é criada com base em regiões da AWS e zonas de disponibilidade. As regiões da AWS As regiões fornecem várias zonas de disponibilidade separadas e isoladas fisicamente, que são conectadas com baixa latência, throughput elevada e redes altamente redundantes. 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 mais altamente disponíveis, tolerantes a falhas e escaláveis que uma ou várias infraestruturas de data center tradicionais.

Para obter mais informações sobre regiões e zonas de disponibilidade da AWS, consulte Infraestrutura global da AWS.

Como o Device Farm está disponível somente na região `us-west-2`, recomendamos enfaticamente 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 no AWS Device Farm

Por ser um serviço gerenciado, o AWS Device Farm é protegido pela segurança da rede global da AWS. Para obter informações sobre serviços de segurança da AWS e como a AWS protege a infraestrutura, consulte Segurança na Nuvem AWS. Para projetar seu ambiente da AWS usando as práticas recomendadas de segurança de infraestrutura, consulte Proteção de infraestrutura em Pilar segurança: AWS Well-Architected Framework.

Você usa AWS chamadas de API publicadas para acessar o Device Farm por meio da rede. Os clientes devem oferecer suporte para:

- 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 você pode 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 as conexões de entrada e saída de 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 sem um terceiro intermediário, externo à AWS.

Todo o tráfego para controladores WebDriver do Selenium deve ser feito por meio do endpoint HTTPS gerado com `createTestGridUrl`.

O recurso de teste do navegador de desktop não é compatível com a configuração de endpoint do Amazon VPC no momento. Você é responsável por garantir que cada instância de teste do Device Farm tenha acesso seguro aos recursos que testa.

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 com Android 4.4.2, o Device Farm não garante que o dispositivo esteja 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 o seu contrato com a AWS.

A Device Farm faz um esforço sincero para impedir que os aplicativos do cliente tomem 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 rodas Python e gemas 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 gems do Ruby, consulte as Práticas de segurança no site RubyGems.
- Para obter informações sobre o pacote de segurança usado por Pipenv e endossado pelo Python Packaging Authority para verificar o gráfico de dependência em busca de vulnerabilidades conhecidas, consulte a Detecção de vulnerabilidades de segurança no GitHub.
- Para obter informações sobre o verificador de dependências Maven do Open Web Application Security Project (OWASP – Projeto Aberto de Segurança em Aplicativos Web), consulte OWASP DependencyCheck no site do OWASP.

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 a AWS for informada 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, a AWS entrará em contato com os clientes afetados, de acordo com o alerta de incidente padrão e as políticas de relatório usadas em todos os serviços da AWS.

Registrar em log e monitorar no Device Farm

Esse serviço é compatível com o AWS CloudTrail, que é um serviço que registra chamadas AWS para o seu Conta da AWS e fornece arquivos de log para um bucket do Amazon S3. Usando as informações coletadas pelo CloudTrail, você pode determinar quais solicitações foram feitas para os

serviços da Serviços da AWS, quem fez a solicitação, quando ela foi feita etc. Para saber mais sobre o CloudTrail, inclusive como ativá-lo e encontrar seus arquivos de log, consulte o Guia do usuário do AWS CloudTrail.

Para obter informações sobre como usar o CloudTrail com o Device Farm, consulte Registro 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 no seu ambiente, trate-as como considerações úteis em vez de requisitos.

- 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 Trabalhar com 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 existe limite quanto ao número de dispositivos que você pode incluir em uma execução de teste. O número de dispositivos, ou trabalhos, nos quais você pode executar testes em paralelo a qualquer momento é igual à simultaneidade no nível da sua conta. A simultaneidade padrão no nível da conta para uso medido no AWS Device Farm é 5. Você pode solicitar um aumento desse número 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.

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 os plug-ins do Device Farm no AWS Labs no GitHub.

Se você for um desenvolvedor Android, também temos um aplicativo de amostra do AWS Device Farm para Android no GitHub. Você pode usar o aplicativo e os testes de exemplo como referência para seus próprios scripts de teste do Device Farm.

Tópicos

- Integração do AWS Device Farm com o plug-in Jenkins CI
- Plug-in Gradle do AWS Device Farm

Integração do AWS Device Farm com o plug-in Jenkins CI

Esse plug-in 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).

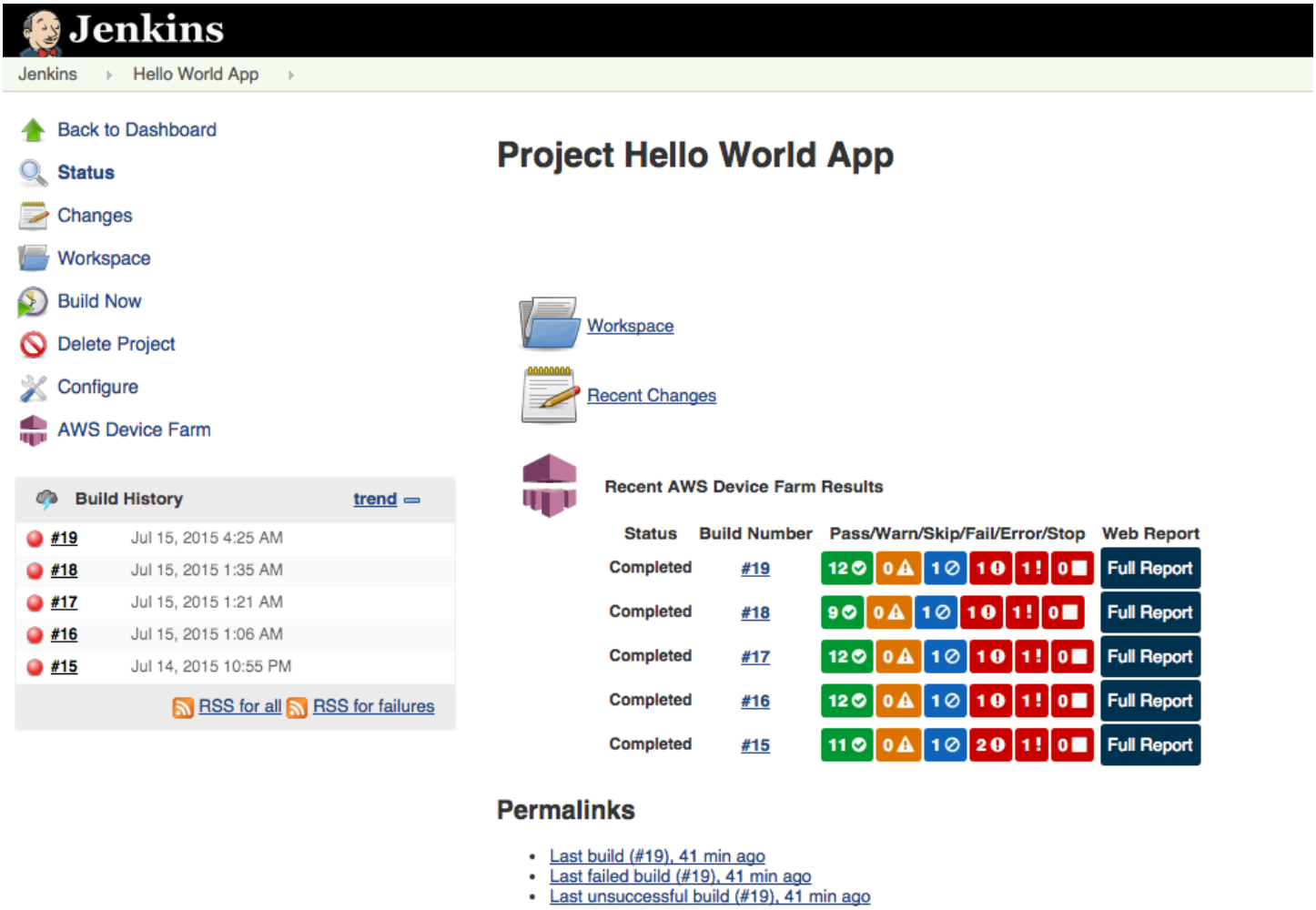
> **Note**
>
> Para fazer download do plug-in Jenkins, acesse o GitHub e siga as instruções em Etapa 1: instalar o plug-in.

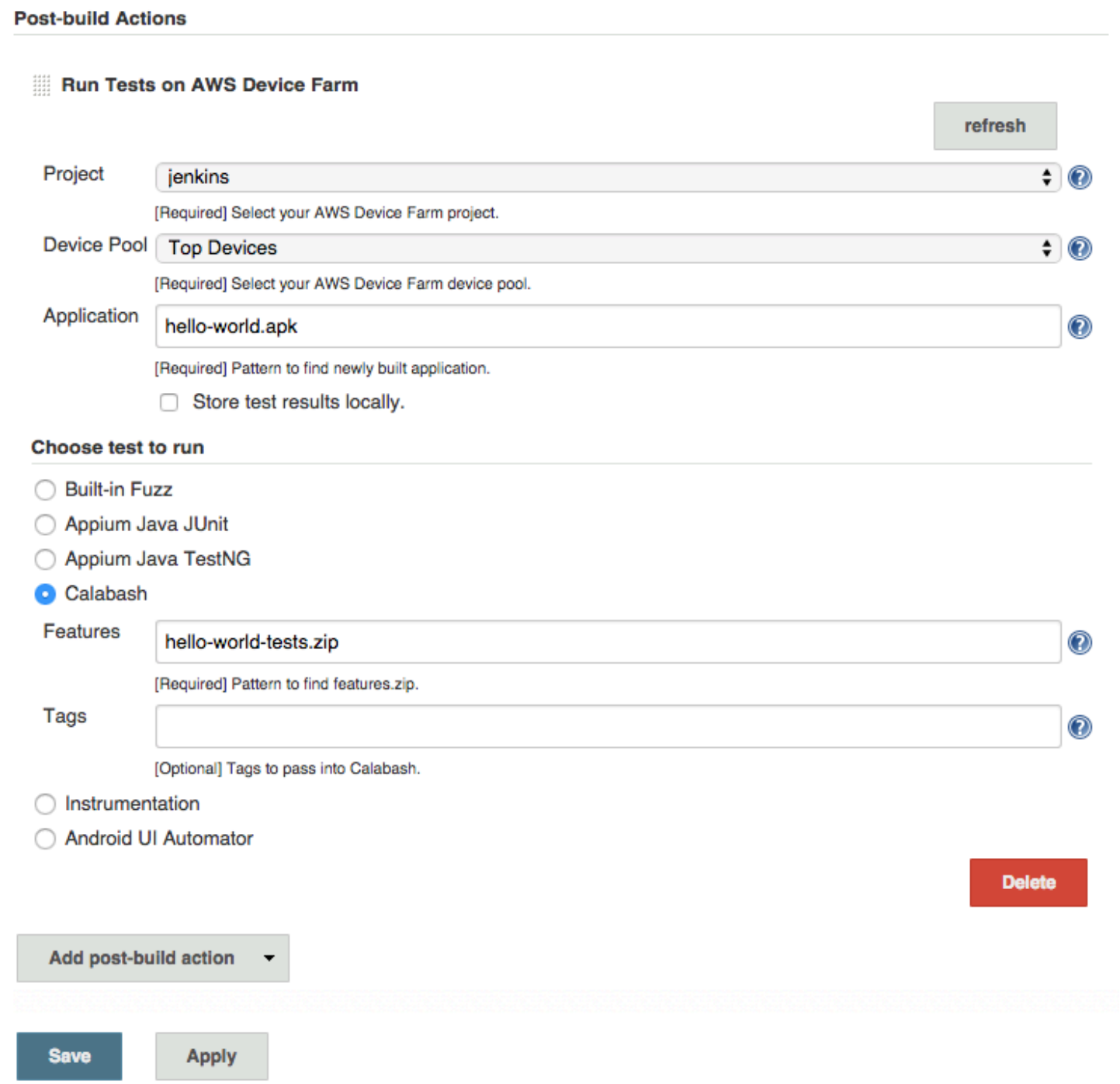
Esta seção contém uma série de procedimentos para configurar e usar o plug-in Jenkins CI com o AWS Device Farm.

Tópicos

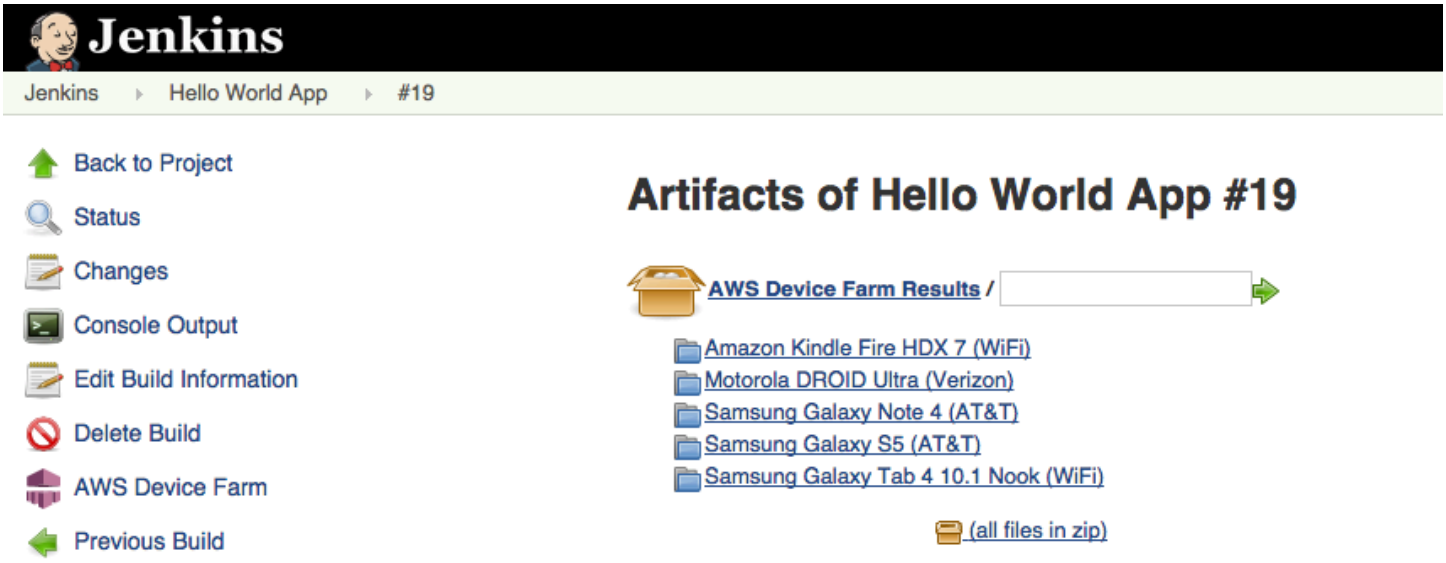
- Etapa 1: instalar o plug-in
- Etapa 2: criar um usuário do AWS Identity and Access Management para o plug-in Jenkins CI
- Etapa 3: instruções sobre configuração inicial
- Etapa 4: usar o plug-in em um trabalho do Jenkins
- Dependências

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:



Etapa 1: instalar o plug-in

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. Procure 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. Faça download do arquivo `hpi` 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 usuário do AWS Identity and Access Management para o plug-in Jenkins CI

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 IAM existente) em sua conta AWS e, em seguida, acesse o Device Farm com esse usuário IAM.

Para criar um novo usuário do IAM, consulte Criação de um usuário do IAM (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 raiz AWS ou o usuário do IAM que você usa para concluir as etapas a seguir deve ter permissão para criar a política de IAM a seguir 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. Escolha Policies (Políticas).
3. Escolha Create Policy (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 o ajude a associar esse usuário IAM ao seu projeto 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 Create Policy (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 Users (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: instruções sobre configuração inicial

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 do AWS Device Farm.
5. Copie suas credenciais de segurança de Etapa 2: criar um usuário do IAM e cole o ID da chave de acesso e a chave de acesso secreta nas respectivas caixas.
6. Escolha Save (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.

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.

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**
>
> Para fazer download do plug-in Gradle, acesse o GitHub e siga as instruções em Criação 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

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 o GitHub e 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 IAM

Etapa 3: Geração de um usuário IAM

O AWS Identity and Access Management (IAM) ajuda você a gerenciar permissões e políticas para trabalhar com os recursos da AWS. Este tópico o orienta na geração de um usuário 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 raiz da AWS para acessar o Device Farm. Em vez disso, crie um novo usuário IAM (ou use um usuário IAM existente) em sua conta da AWS e acesse o Device Farm com esse usuário IAM.

> **Note**
>
> A conta raiz da AWS ou o usuário do IAM que você usa para concluir as etapas a seguir deve ter permissão para criar a política de IAM a seguir 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 Users (Usuários).
3. Escolha Create New Users (Criar novos usuários).
4. Digite o nome de usuário de sua escolha.

   Por exemplo, **`GradleUser`**.
5. Escolha Create (Criar).
6. Escolha Download Credentials (Fazer download de credenciais) e as salve em um local onde você possa recuperá-las facilmente depois.
7. Escolha Close (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 Criação de um usuário IAM (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 Trabalhar com 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 Trabalho com tipos de teste 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 oferece suporte para Appium Java JUnit e 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 ter mais informações, consulte Integrado: Fuzz (Android e iOS).

Instrumentação

O Device Farm oferece suporte para instrumentação (JUnit, Espresso, Robotium ou qualquer outro teste baseado em instrumentação) para Android. Para ter mais informações, consulte Trabalhar com instrumentação para Android e AWS Device Farm.

Ao executar um teste de instrumentação no Gradle, o Device Farm usa o arquivo `.apk` gerado a partir do diretório androidTest como a fonte dos seus testes.

```
    instrumentation {
```

```
        filter "test filter per developer docs" // optional

    }
```

Dependências

Tempo de execução

- O plug-in Gradle do Device Farm 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

Histórico do documento

A tabela a seguir descreve as mudanças importantes na documentação desde a última versão deste guia.

Alteração	Descrição	Alterado em
Suporte AL2	O Device Farm agora é compatível com o ambiente de teste AL2 para Android. Saiba mais sobre o 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 Device Farm agora suporta a estrutura Polaris.	28 de julho de 2021
Suporte ao Python 3	O Device Farm agora suporta 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 a proteção de serviços da AWS mais simples e abrangente, foi criada uma nova seção sobre segurança. Para ler mais, consulte Segurança em AWS Device Farm	27 de março de 2020

Alteração	Descrição	Alterado em
a marcação de recursos da AWS.	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 o XCTest	Para execuções de teste que usam a estrutura XCTest, 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 Trabalho com tipos de teste 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 Trabalho com tipos de teste 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 registro 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 Trabalhar com ambientes de teste personalizados.	16 de agosto de 2018
Suporte para o uso do Device Farm como um provedor de teste AWS CodePipeline	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 Uso do AWS Device Farm em um estágio de teste do CodePipeline.	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 Amazon VPC para testar aplicativos privados e criar sessões de depuração remota. Saiba mais sobre Trabalhando com 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 Definir 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 Simule 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 Trabalhar com acesso remoto.	19 de abril de 2016
Autoatendimento para slots de dispositivo	Você já pode comprar slots de dispositivos usando o AWS Management Console, a AWS Command Line Interface ou a API. Saiba mais sobre como Compre um slot de dispositivo no Device Farm.	22 de março de 2016

Alteração	Descrição	Alterado em
Como interromper execuções de teste	Você já pode interromper execuções de teste usando o AWS Management Console, a AWS Command Line Interface ou a API. Saiba mais sobre como Interromper uma execução no AWS Device Farm.	22 de março de 2016
Novos tipos de teste do XCTest UI	Agora você pode executar testes personalizados do XCTest UI em aplicativos iOS. Saiba mais sobre o tipo de teste XCTest UI.	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 Trabalho com tipos de teste no AWS Device Farm.	19 de janeiro de 2016
Tipos de teste de aplicativos web	Você já pode executar testes personalizados do Appium Java JUnit e TestNG em aplicativos web. Saiba mais sobre Trabalhando com testes de aplicativos web no AWS Device Farm.	19 de novembro de 2015
Plug-in para 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 o teste de dispositivos iOS e a execução de testes de iOS (incluindo o XCTest) Trabalho com tipos de teste no AWS Device Farm.	4 de agosto de 2015
Lançamento público inicial	Esta é a versão pública inicial do Guia do desenvolvedor do AWS Device Farm.	13 de julho de 2015

Glossário do AWS

Para obter a terminologia mais recente da AWS, consulte o glossário da AWS na Referência do Glossário da AWS.

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.