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 aplicações que você pode usar para testar e interagir com suas aplicações Android, iOS e web em telefones e tablets reais e físicos hospedados pela Amazon Web Services (AWS).

Existem duas maneiras principais de usar o Device Farm:

- Acesse remotamente um dispositivo a partir do seu computador local, de forma interativa em seu navegador da web ou testando-o automaticamente usando o Appium a partir de um cliente local.
- Execute testes de aplicativos automaticamente usando o ambiente gerenciado de execução de testes do Device Farm.

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

Acesso remoto

O acesso remoto permite que você interaja com um dispositivo por meio do seu navegador da web em tempo real. O acesso remoto também permite que você execute testes Appium do seu cliente local em dispositivos Device Farm remotos usando um endpoint Appium gerenciado.

A interação em tempo real com um dispositivo pode ser útil em vários cenários, como testes manuais de aplicativos, reprodução de bugs em um dispositivo específico, verificação da renderização visual do aplicativo em diferentes tipos de tela e sequências de instalação e atualização do aplicativo. O endpoint Appium totalmente gerenciado do Device Farm permite que você desenvolva, teste e depure seus testes do Appium, fornecendo feedback rápido.

O endpoint Appium suporta qualquer linguagem de sua escolha, qualquer IDE local, depuração ao vivo com pontos de interrupção, vídeo e registros ao vivo e ferramentas como o Appium Inspector. Você pode executar testes quantas vezes quiser no mesmo dispositivo durante sua sessão de acesso remoto com um limite de 150 minutos.

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

Teste automatizado de aplicações

O Device Farm permite que você execute testes automatizados em vários dispositivos em paralelo, fazendo o upload do aplicativo e dos testes. Os testes são executados automaticamente em um ambiente totalmente gerenciado em hosts de teste nos quais você pode configurar um arquivo de especificação de teste. O ambiente usa os hosts de teste do Device Farm, então você não precisa se preocupar em provisionar sua própria infraestrutura para a execução de testes. Os hosts e dispositivos de teste podem se conectar com segurança à sua VPC para acessar seus endpoints privados.

À medida que os testes são concluídos, é gerado um relatório de teste que contém resultados de alto nível, registros de baixo nível, capturas de tela e seus artefatos de teste.

O Device Farm suporta testes de aplicativos Android e iOS nativos e híbridos. Para obter mais informações sobre tipos de teste compatíveis, consulte Frameworks de teste e testes integrados no AWS Device Farm.

Terminologia

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

grupo de dispositivos

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

trabalho

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

medição

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

project

: Um espaço de trabalho lógico que contém execuções, uma para cada teste de um único aplicativo em um ou mais dispositivos. Você pode usar projetos para organizar os espaços

de trabalho da forma que você escolher. Por exemplo, você pode ter um projeto por título de aplicativo ou um projeto por plataforma. Você pode criar quantos projetos necessitar.

relatório

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

run

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

sessão

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

pacote

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

teste

Um caso de teste específico em um pacote de testes.

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

Configurar

Para usar o Device Farm, consulte Configurar.

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

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

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

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

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

Para obter mais informações, consulte Creating an IAM User (Console de gerenciamento da AWS).

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

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

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

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

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

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

Próxima etapa

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

Conceitos básicos do Device Farm

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

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

Tópicos

Pré-requisitos

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

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

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

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

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

Etapa 1: Fazer login no console do

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

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

Etapa 2: criar um projeto

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

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

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

Etapa 3: Criar e iniciar uma execução

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

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

testando uma aplicação Android (.apk). Selecione Aplicação iOS se estiver testando uma aplicação iOS (.ipa).

4. Em Selecionar aplicação, na seção Opções de seleção de aplicações, escolha Selecionar aplicação de exemplo fornecida pelo Device Farm se não tiver uma aplicação disponível para teste. Se você estiver trazendo sua própria aplicação, selecione Fazer upload da própria aplicação e escolha o arquivo da sua aplicação. Se você estiver fazendo upload de um aplicativo iOS, certifique-se de escolher iOS device (Dispositivo iOS), e não um simulador.
5. Na página Configurar teste, na seção Selecionar framework de teste, escolha um dos frameworks de teste ou pacotes de testes integrados. Para ter mais informações sobre cada opção, consulte Frameworks de teste e testes integrados.
   - Se você ainda não tiver empacotado seus testes para o Device Farm, escolha Integrado: fuzz para executar um conjunto de testes padrão integrado. Você pode manter os valores padrão para Contagem de eventos, Restrição de eventos e Semente aleatória. Para obter mais informações, consulte the section called “Integrado: Fuzz (Android e iOS)”.
   - Se você tiver um pacote de teste de uma das estruturas de teste compatíveis, escolha a estrutura de teste correspondente e, em seguida, faça o upload do arquivo que contém seus testes.
6. Em Selecionar dispositivos, escolha Usar grupo de dispositivos e Principais dispositivos.
7. (Opcional) Para adicionar configurações adicionais, abra o menu suspenso Configuração adicional. Nesta seção, é possível realizar qualquer um destes procedimentos:
   - Para fornecer outros dados para o Device Farm usar durante a execução, ao lado de Adicionar dados extras, selecione Escolher arquivo e, em seguida, navegue até o arquivo .zip que contém os dados e escolha-o.
   - Para instalar uma aplicação adicional para o Device Farm usar durante a execução, ao lado de Instalar outras aplicações, selecione Escolher arquivo e, em seguida, procure e escolha o arquivo .apk ou .ipa que contém a aplicação. Repita isso para outros aplicativos que você deseja instalar. Você pode alterar a ordem de instalação arrastando e soltando os aplicativos depois de fazer upload deles.
   - Para especificar se Wi-Fi, Bluetooth, GPS ou NFC estará habilitado durante a execução, ao lado de Set radio states (Definir estados de rádio), selecione as caixas apropriadas.
   - Para predefinir a latitude e a longitude do dispositivo para a execução, ao lado de Device location (Local do dispositivo), insira as coordenadas.
   - Para predefinir a localidade da execução, em Localidade do dispositivo, escolha a localidade.
   - Selecione Habilitar a gravação de vídeo para gravar vídeos durante o teste.

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

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

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

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

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

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

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

Etapa 4: visualizar os resultados da execução

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

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

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

Para obter mais informações, consulte Visualizar relatórios de testes no Device Farm.

Próximas etapas

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

Comprar um slot de dispositivo no Device Farm

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

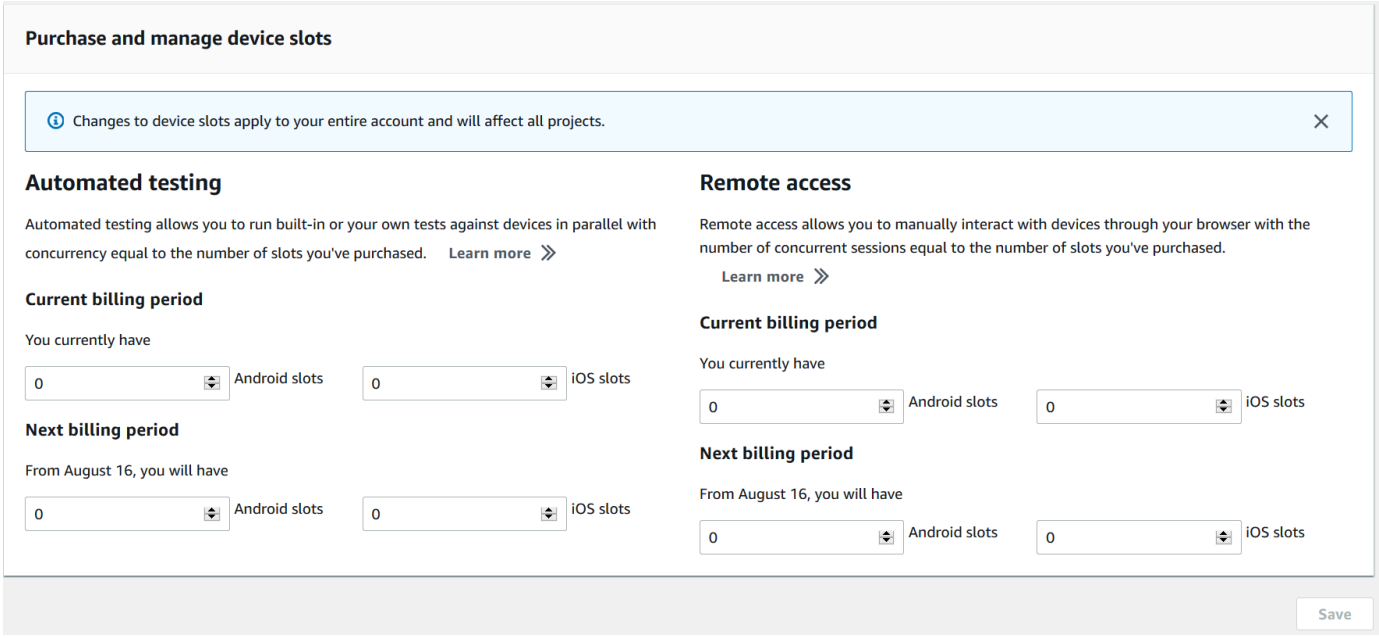
Comprar slots de dispositivos (console)

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

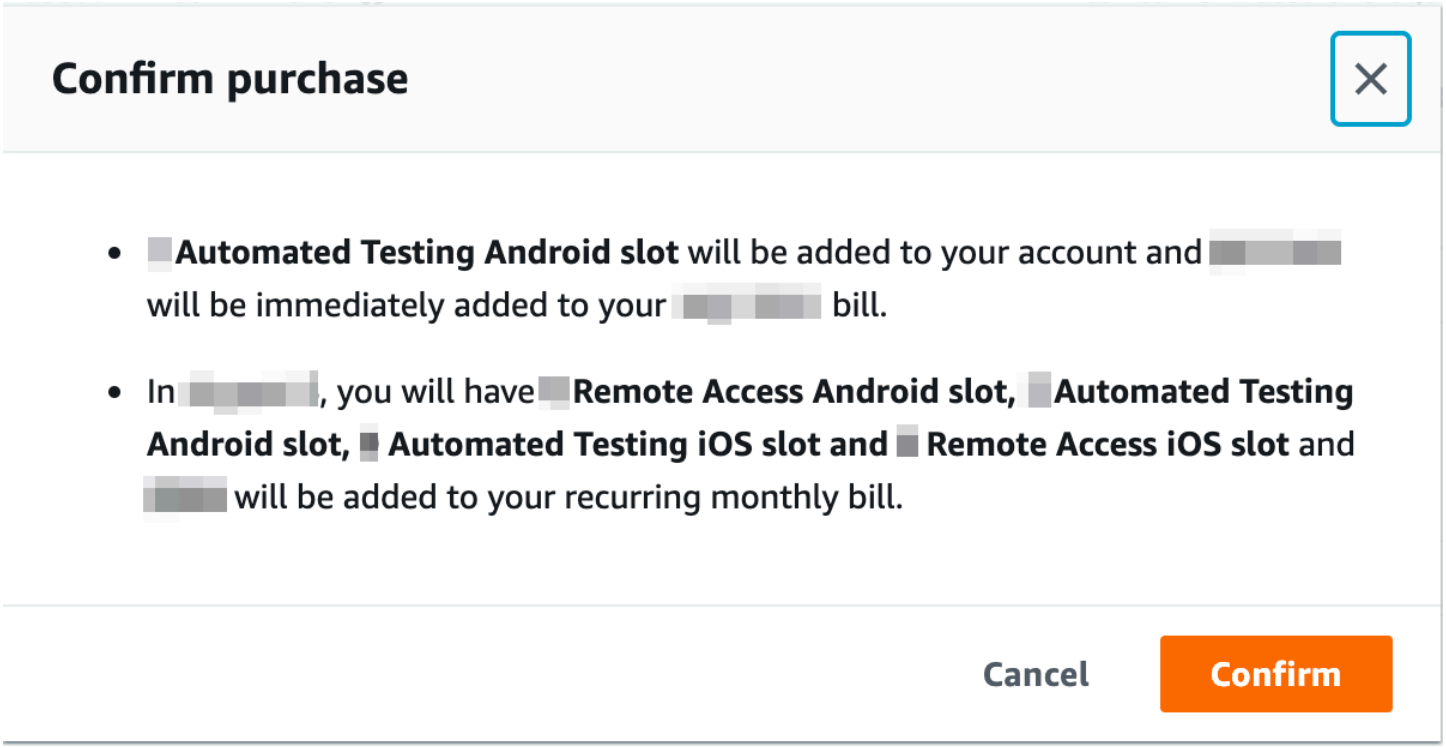
   Conforme você altera o valor do slot, o texto é atualizado dinamicamente com o valor do faturamento. Para acessar mais informações, consulte Preços do AWS Device Farm.

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

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



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



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

Comprar um slot de dispositivo (AWS CLI)

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

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

```
{
    "accountSettings": {
        "maxSlots": {
            "GUID": 1,
            "GUID": 1,
            "GUID": 1,
            "GUID": 1
        },
        "unmeteredRemoteAccessDevices": {
            "ANDROID": 0,
            "IOS": 0
        },
        "maxJobTimeoutMinutes": 150,
        "trialMinutes": {
            "total": 1000.0,
            "remaining": 954.1
        },
        "defaultJobTimeoutMinutes": 150,
        "awsAccountNumber": "AWS-ACCOUNT-NUMBER",
        "unmeteredDevices": {
            "ANDROID": 0,
            "IOS": 0
        }
    }
}
```

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

```
{
```

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

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

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

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

Você deve ver uma saída semelhante a:

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

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

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

```
}
```

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

Comprar um slot de dispositivo (API)

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

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

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

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

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

Cancelar um slot de dispositivo no Device Farm

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

Para acessar mais informações sobre slots de dispositivos, consulte Comprar um slot de dispositivo no Device Farm.

Cancelar um slot de dispositivo (console)

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

Cancelar um slot de dispositivo (AWS CLI)

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

Cancelar um slot de dispositivo (API)

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

Conceitos do AWS Device Farm

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

Esta seção descreve conceitos importantes do Device Farm.

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

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

Suporte de dispositivos no AWS Device Farm

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

Tópicos

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

Dispositivos compatíveis

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

Grupos de dispositivos

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

Dispositivos privados

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

Marcas de dispositivo

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

Slots para dispositivo

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

Existem dois tipos de slots de dispositivos:

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

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

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

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

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

Aplicações de dispositivos pré-instalados

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

Recursos dos dispositivos

Todos os dispositivos têm conectividade 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 e o Google Chrome estão instalados em dispositivos Android.

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

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

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

Ambiente de teste personalizado

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

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

Para obter mais informações, consulte Uploading a Custom Test Spec Using the AWS CLI e 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 da aplicação, com um conjunto específico de testes, a ser executada em um conjunto específico de dispositivos. Uma

execução produz um relatório que contém informações sobre os resultados da execução. A execução contém um ou mais trabalhos.

Tópicos

Configuração da execução

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, dados extras (contidos em um arquivo.zip) e aplicativos auxiliares (aplicativos que devem ser instalados antes do aplicativo a ser testado). No Android, algumas configurações adicionais podem ser alteradas, como localidade e estados de rádio (Bluetooth, GPS, NFC e Wi-Fi).

Retenção de arquivos de execução

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

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

Estado do dispositivo de execução

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

Execuções paralelas

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

Configurar o tempo limite de execução

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

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

Anúncios nas execuções

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

Mídias nas execuções

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

Tarefas comuns nas execuções

Para obter mais informações, consulte Criar uma execução de teste no Device Farm e Execuções de teste no AWS Device Farm.

Aplicações no AWS Device Farm

As seções a seguir contêm informações sobre comportamentos da aplicação no Device Farm.

Tópicos

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

Instrumentação de aplicações

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

Nova assinatura de aplicações nas execuções

Para aplicações 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 a aplicação. Se você fornecer dados auxiliares, o Device Farm os adicionará ao pacote do aplicativo antes de instalá-lo, para que o auxiliar exista no sandbox da aplicação. A nova assinatura do aplicativo elimina direitos como App Group, Associated Domains, Game Center, HealthKit, HomeKit, Wireless Accessory Configuration, In-App Purchase, Inter-App Audio, Apple Pay, Push Notifications e VPN Configuration & Control.

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

Aplicações ofuscadas nas execuções

Em aplicações para Android, se a aplicação estiver ofuscada, você ainda poderá testá-la 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 e executar testes na aplicação.

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, logs de testes e dispositivos, capturas de tela e dados de desempenho.

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

Logs nos relatórios

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

Tarefas comuns relacionadas aos relatórios

Para obter mais informações, consulte Visualizar relatórios de testes 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. Isso inclui a interação manual em um navegador da web e a execução de testes do Appium a partir de um cliente local em relação ao dispositivo remoto. Os desenvolvedores podem reproduzir problemas com o aplicativo ou com os testes do Appium em um dispositivo específico para isolar e resolver problemas.

Tópicos

Dispositivos compatíveis com acesso remoto

O Device Farm é compatível com uma série de dispositivos Android e iOS exclusivos e populares. A lista de dispositivos disponíveis cresce à medida que novos dispositivos entram no mercado. O

console do Device Farm exibe a lista atual de dispositivos Android e iOS disponíveis para acesso remoto. Para obter mais informações, consulte Suporte de dispositivos no AWS Device Farm.

Retenção de arquivos de sessão

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

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

Instrumentação de aplicações

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

Nova assinatura de aplicações nas sessões

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

Aplicações ofuscadas nas sessões

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

Projetos no AWS Device Farm

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

Você pode usar o console do AWS Device Farm, a 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 Configurar.

Criar um projeto (console)

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e, em seguida, Projetos.
3. Selecione New project (Novo projeto).
4. Insira um nome para o projeto. Opcionalmente, você pode fornecer um ou mais dos parâmetros abaixo e escolher Enviar.

Configurações da Virtual Private Cloud (VPC)

Selecione uma VPC, sub-redes e grupo de segurança a serem aplicados ao dispositivo em teste e ao host de teste emparelhado. Esse recurso só é compatível com dispositivos privados. Consulte VPC-ENI no AWS Device Farm para obter mais informações.

ARN da função de execução

Uma função do IAM a ser assumida pelo executor de testes em ambientes de teste personalizados. Para obter mais informações, consulte Acesse os recursos da AWS usando uma função de execução do IAM.

Variáveis de ambiente

Uma ou mais variáveis a serem inseridas no ambiente do processo do executor de execução do teste. Os nomes das variáveis que começam com “DEVICEFARM_” são reservados para uso do serviço. Não recomendamos armazenar valores confidenciais nessas variáveis de ambiente e, em vez disso, sugerimos o uso de uma função de execução do IAM para buscar esses valores do AWS Secrets Manager durante o teste.

Criar um projeto (AWS CLI)

- Execute create-project especificando o nome do projeto.

  Exemplo:

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

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

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

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

Criar um projeto (API)

- Chame a API `CreateProject`.

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

Visualizar a lista de projetos no AWS Device Farm

Você pode usar o console do AWS Device Farm, a 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 Teste para dispositivos móveis e escolha Projetos.
   - Para projetos de teste de navegadores de desktop, no menu de navegação do Device Farm, escolha Teste para navegadores de área de trabalho e Projetos.

Visualizar a lista de projetos (AWS CLI)

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

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

Para obter informações sobre como usar o Device Farm com a AWS CLI, consulte AWS CLIReferência do .

Visualizar a lista de projetos (API)

- Para visualizar a lista de projetos, chame 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.

Execuções de teste no AWS Device Farm

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

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

Tópicos

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.

Criar uma execução de teste (console)

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. No painel de navegação, escolha Teste para dispositivos móveis e Projetos.
3. Se já tiver um projeto, você poderá fazer upload dos testes para ele. Caso contrário, selecione Novo projeto, insira um Nome do projeto e escolha Criar.
4. Abra o projeto e escolha Criar execução.
5. (Opcional) Em Configurações de execução, na seção Nome da execução, insira um nome para a execução. Se nenhum nome for fornecido, o console do Device Farm chamará sua execução de “Minha execução do Device Farm” por padrão.
6. (Opcional) Em Configurações de execução, na seção Tempo limite do trabalho, você pode especificar o tempo limite de execução para sua execução de teste. Se você estiver usando um número ilimitado de slots de testes, confirme se Não medido está selecionado em Método de cobrança.
7. Em Configurações de execução, na seção Tipo de execução, selecione seu tipo de execução. Selecione Aplicação Android se você não tiver uma aplicação pronta para teste ou se estiver testando uma aplicação Android (.apk). Selecione Aplicação iOS se estiver testando uma aplicação iOS (.ipa). Selecione Aplicativo web se quiser testar aplicativos web.
8. Em Selecionar aplicação, na seção Opções de seleção de aplicações, escolha Selecionar aplicação de exemplo fornecida pelo Device Farm se você não tiver uma aplicação disponível para teste. Se você estiver trazendo sua própria aplicação, selecione Fazer upload da própria aplicação e escolha o arquivo da sua aplicação. Se você estiver fazendo upload de um aplicativo iOS, certifique-se de escolher iOS device (Dispositivo iOS), e não um simulador.
9. Em Configurar teste, escolha um dos frameworks de teste disponíveis.

> **Note**
>
> Se não tiver testes disponíveis, escolha Integrado: fuzz para executar um pacote integrado padrão de testes. Se você escolher Integrado: fuzz e as caixas Contagem de

eventos, Limite de eventos e Propagação aleatória forem exibidas, poderá alterar ou manter os valores.

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

10. Se você não escolheu Integrado: Fuzz, selecione Escolher arquivo em Selecionar pacote de testes. Procure e escolha o arquivo que contém os testes.
11. Para seu ambiente de teste, escolha Executar teste em nosso ambiente padrão ou Executar teste em um ambiente personalizado. Para obter mais informações, consulte Ambientes de teste no AWS Device Farm.
12. Se você estiver usando um ambiente de teste personalizado, poderá fazer o seguinte:
    - Se você quiser editar a especificação de teste padrão em um ambiente de teste personalizado, escolha Editar para atualizar a especificação YAML padrão.
    - Se você fizer alterações na especificação de teste, escolha Salvar como novo para atualizar.
    - Você pode configurar variáveis de ambiente. As variáveis fornecidas aqui terão precedência sobre qualquer uma que possa ser configurada no projeto principal.
13. Em Selecionar dispositivos, siga um destes procedimentos:
    - 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.
    - Escolha Selecionar dispositivos manualmente e selecione os dispositivos nos quais você deseja executar os testes. Essa configuração não será salva.

    Para obter mais informações, consulte Suporte de dispositivos no AWS Device Farm.
14. (Opcional) Para adicionar configurações adicionais, abra o menu suspenso Configuração adicional. Nesta seção, é possível realizar qualquer um destes procedimentos:
    - Para fornecer um ARN da função de execução ou substituir um configurado no projeto principal, use o campo ARN da função de execução.

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

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

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

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

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

na tabela de execução. Cada dispositivo na execução também começará

com o ícone pendente e, em seguida, mudará para o ícone em execução

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

Se você quiser interromper a execução do teste, consulte 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: escolher um grupo de dispositivos

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

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

   Exemplo:

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

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

   ```
   {
       "devicePools": [
           {
               "rules": [
                   {
                       "attribute": "ARN",
                       "operator": "IN",
                       "value": "[\"arn:aws:devicefarm:us-west-2::device:example1\",
   \"arn:aws:devicefarm:us-west-2::device:example2\",\"arn:aws:devicefarm:us-
   west-2::device:example3\"]"
                   }
               ],
               "type": "CURATED",
               "name": "Top Devices",
               "arn": "arn:aws:devicefarm:us-west-2::devicepool:example",
               "description": "Top devices"
   ```

```
        },
        {
            "rules": [
                {
                    "attribute": "PLATFORM",
                    "operator": "EQUALS",
                    "value": "\"ANDROID\""
                }
            ],
            "type": "PRIVATE",
            "name": "MyAndroidDevices",
            "arn": "arn:aws:devicefarm:us-west-2:605403973111:devicepool:example2"
        }
    ]
}
```

2. Escolha um grupo de dispositivos e anote o ARN.

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

Etapa 3: fazer upload do arquivo da aplicação

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

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

Para obter mais informações, consulte create-upload.

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

   Este exemplo cria um upload para um aplicativo Android:

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

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

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

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

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

Para obter mais informações, consulte Carregar objetos usando presigned URLs no Guia do usuário do Amazon Simple Storage Service.

4. Para verificar o status de upload do aplicativo, execute get-upload e especifique o ARN de upload do aplicativo.

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

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

```
{
    "upload": {
        "status": "SUCCEEDED",
        "name": "MyAndroid.apk",
        "created": 1535732625.964,
        "url": "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/
ExampleURL",
```

```
        "type": "ANDROID_APP",
        "arn": "arn:aws:devicefarm:us-west-2:123456789101:upload:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE",
        "metadata": "{"valid": true}"
    }
}
```

Etapa 4: fazer upload do pacote de scripts de teste

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
{
    "uploads": [
        {

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

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

   Exemplo:

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

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

   ```
   curl "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/ExampleURL" >
    MyTestSpec.yml
   ```
5. Você pode editar a especificação de teste padrão para atender aos requisitos de teste e, em seguida, usar a especificação de teste modificada em execuções de teste futuras. Ignore esta etapa para usar a especificação de teste padrão no estado em que ela se encontra em um ambiente de teste personalizado.

6. Para criar um upload da especificação de teste personalizada, execute create-upload especificando o nome e o tipo da especificação de teste e o ARN do projeto.

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

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

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

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

7. Anote o ARN do upload da especificação de teste e o URL pré-assinado.

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

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

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

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

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

   ```
   {
   ```

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

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

Etapa 6: programar uma execução de teste

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

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

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

Para programar uma execução em um ambiente de teste padrão

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

  Exemplo:

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

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


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


Para obter mais informações, consulte schedule-run.

Para programar uma execução em um ambiente de teste personalizado

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

  Exemplo:

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

Para verificar o status da execução de teste

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

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

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

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 obter mais informações, consulte Etapa 5: (opcional) fazer upload de sua especificação de teste personalizada e `CreateUpload`.

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

Próximas etapas

No console do Device Farm, o ícone de relógio

muda para um ícone de resultado, como sucesso

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

Para usar o relatório, siga as instruções em Visualizar relatórios de testes no Device Farm.

Definir o tempo limite para execuções de teste no AWS Device Farm

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

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

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

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

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

Pré-requisitos

1. Siga as etapas em Configurar.
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 Teste para dispositivos móveis e, em seguida, Projetos.
3. Se você já tiver um projeto, selecione-o na lista. Caso contrário, escolha Novo projeto, insira um nome para o projeto e, em seguida, selecione Enviar.
4. Escolha Configurações do projeto.
5. Na guia Geral, em Tempo limite de execução, insira um valor ou use a barra deslizante.
6. Escolha Salvar.

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

Definir o tempo limite de execução para uma execução de teste

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e, em seguida, Projetos.
3. Se você já tiver um projeto, selecione-o na lista. Caso contrário, escolha Novo projeto, insira um nome para o projeto e, em seguida, selecione Enviar.
4. Escolha Criar uma nova execução.
5. Siga as etapas para escolher um aplicativo, configurar o teste, selecionar os dispositivos e especificar um estado para o dispositivo.
6. Em Analisar e iniciar a execução, para Definir tempo limite de execução, insira um valor ou use a barra deslizante.
7. Escolha Confirmar e iniciar a execução.

Simular 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 e web no Device Farm. Por exemplo, você pode simular conectividade com a Internet intermitente ou com perdas.

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

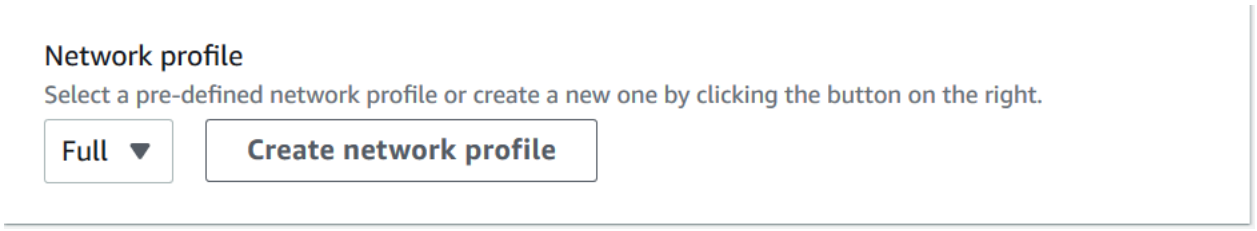
Tópicos

Configurar a modelagem de rede ao programar uma execução de teste

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 sua aplicação e, em seguida, selecione Próximo.
3. Configure seu teste e, em seguida, escolha Próximo.
4. Selecione seus dispositivos e, em seguida, escolha Próximo.
5. Na seção Configurações de localização e rede, escolha um perfil de rede ou selecione Criar perfil de rede para criar o seu próprio perfil.

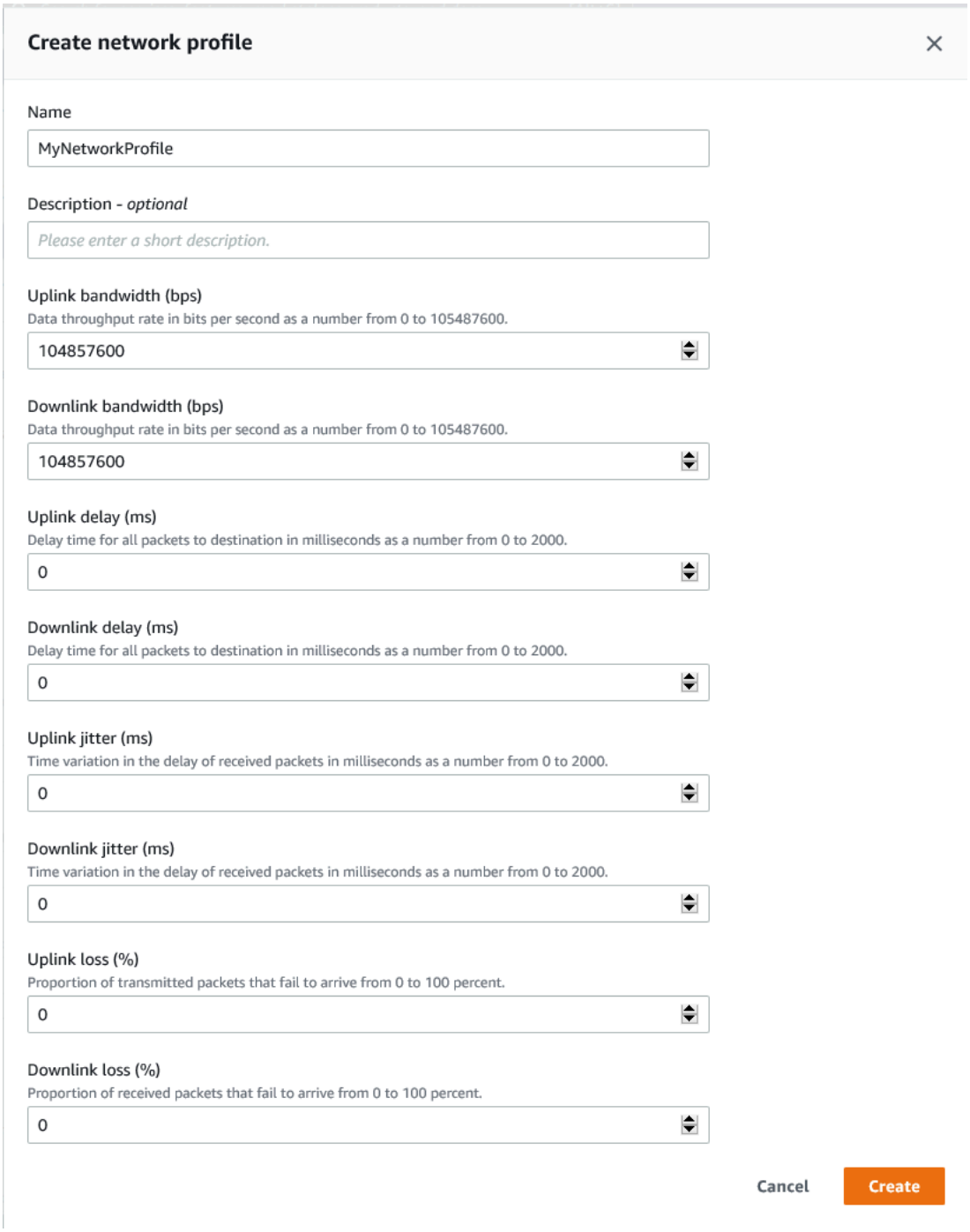


6. Escolha Próximo.
7. Analise e inicie a execução de teste.

Criar um perfil de rede

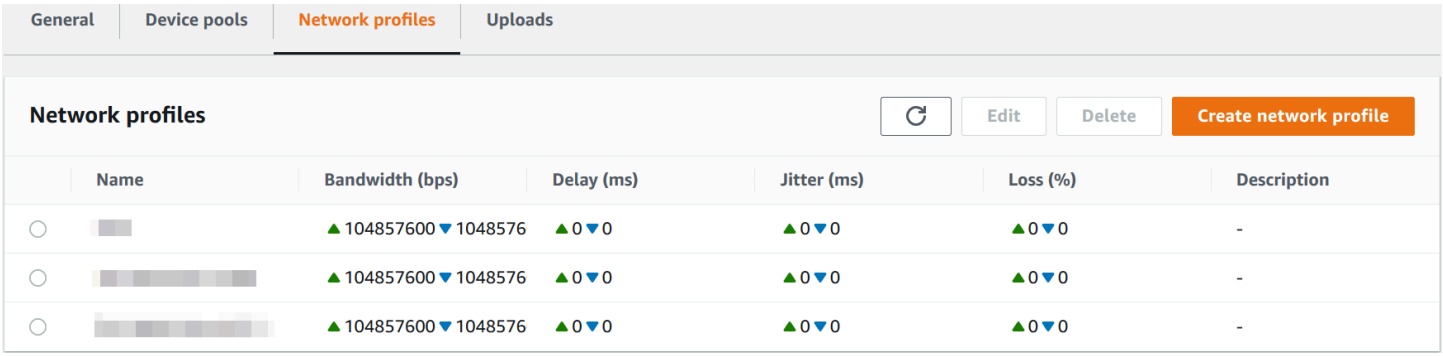
Ao criar uma execução de teste, você pode criar um perfil de rede.

1. Escolha Criar perfil de rede.



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

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



Alterar as condições da rede durante o teste

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

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

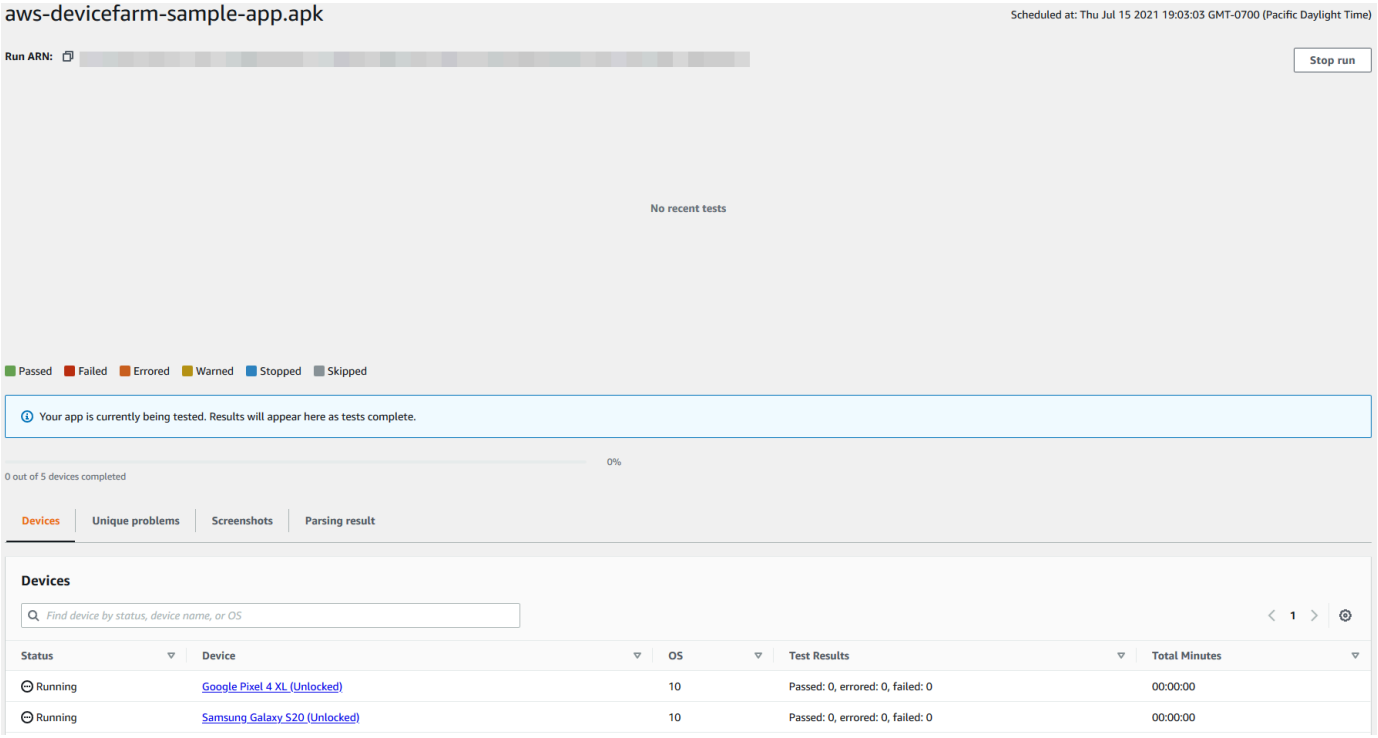
Tópicos

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

Interromper uma execução (console)

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

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

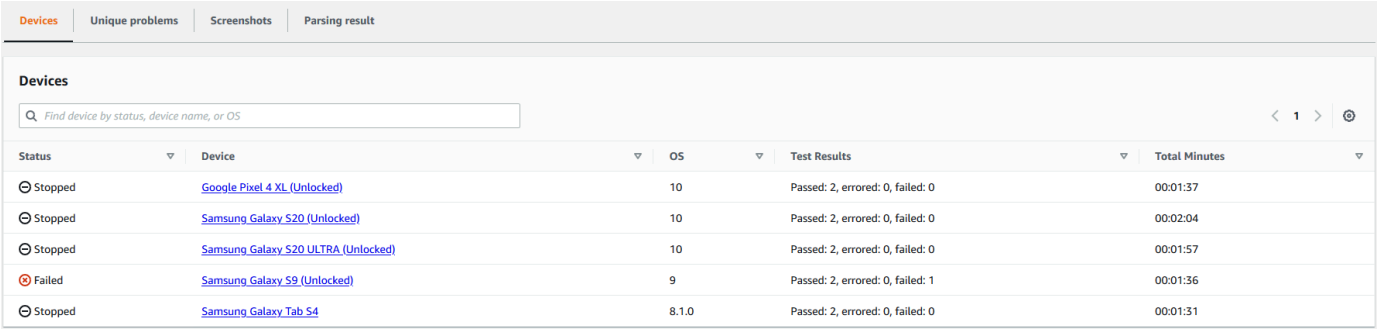


5. Escolha Interromper a execução.

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

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

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



Interromper uma execução (AWS CLI)

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

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

Você deve ver uma saída semelhante a:

```
{
    "run": {
        "status": "STOPPING",
        "name": "Name of your run",
        "created": 1458329687.951,
        "totalJobs": 7,
        "completedJobs": 5,
        "deviceMinutes": {
            "unmetered": 0.0,
            "total": 0.0,
            "metered": 0.0
        },
        "platform": "ANDROID_APP",
        "result": "PENDING",
        "billingMethod": "METERED",
        "type": "BUILTIN_EXPLORER",
        "arn": "myARN",
        "counters": {
            "skipped": 0,
            "warned": 0,
            "failed": 0,
            "stopped": 0,
            "passed": 0,
```

```
            "errored": 0,
            "total": 0
        }
    }
}
```

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

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

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

Interromper uma execução (API)

- Chame a StopRunoperação para a execução do teste.

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

Visualizar uma lista de execuções no AWS Device Farm

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

Tópicos

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

Visualizar uma lista de execuções (console)

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

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

Visualizar uma lista de execuções (AWS CLI)

- Execute o comando list-runs.

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

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

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

a. Escolha Criar grupos dinâmicos de dispositivos.

b. Escolha Adicionar uma regra.

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

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

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

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

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

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

g. Em Máximo de dispositivos, insira o número de dispositivos que você deseja usar no seu grupo de dispositivos. Se você não inserir o número máximo de dispositivos, o Device Farm

escolherá todos os dispositivos da frota que correspondam às regras criadas. Para evitar cobranças adicionais, defina esse número como um valor que corresponda aos requisitos reais de execução paralela e variedade de dispositivos.

h. Para excluir uma regra, escolha Remove regra.

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

   a. Escolha Criar grupos estáticos de dispositivos.

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

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

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

7. Escolha Criar.

Criar um grupo de dispositivos (AWS CLI)

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

- Execute o comando create-device-pool.

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

Criar um grupo de dispositivos (API)

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

- Chame a API `CreateDevicePool`.

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

Analisar os resultados dos testes no AWS Device Farm

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

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

Tópicos

- Visualizar relatórios de testes no Device Farm
- Baixar artefatos no Device Farm

Visualizar relatórios de testes no Device Farm

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

Tópicos

Pré-requisitos

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 obter mais informações, consulte Status dos resultados do teste do Device Farm.

Visualizar relatórios

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

Tópicos

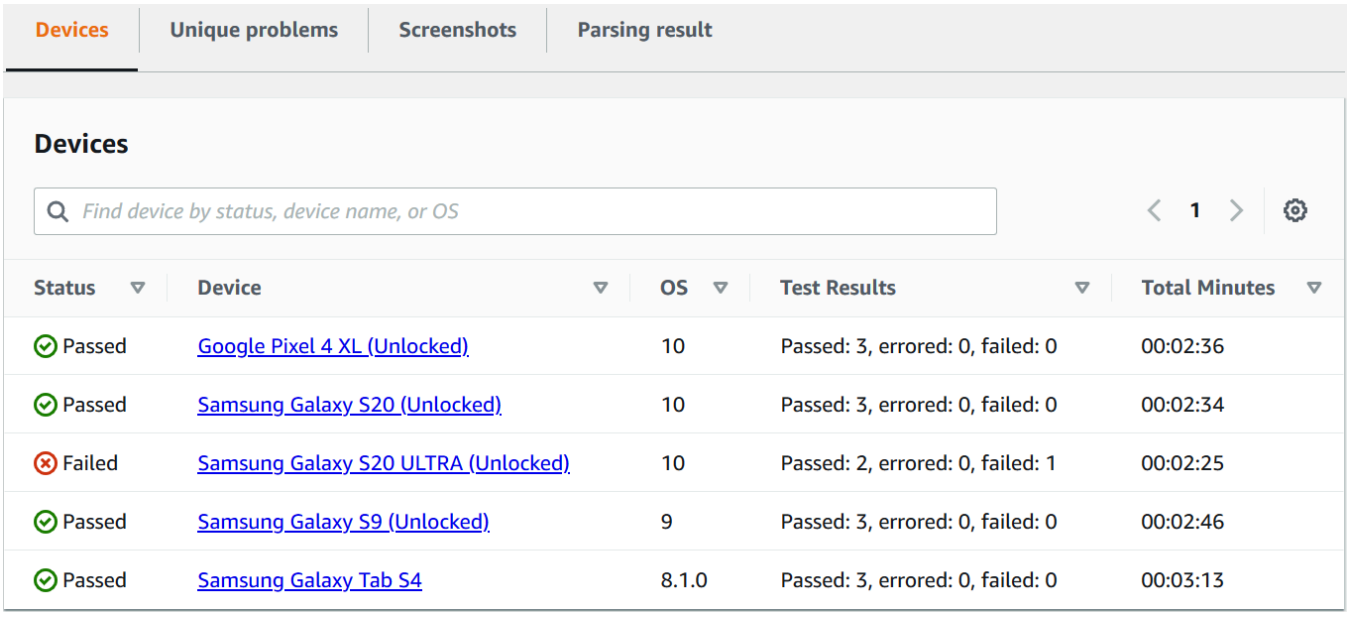
- Visualizar relatórios do conjunto de testes
- Visualizar relatórios de teste
- Visualizar informações de log de um problema, dispositivo, conjunto ou teste em um relatório

Visualizar a página de resumo da execução de teste

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

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

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



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

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

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

Visualizar relatórios de problemas exclusivos

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

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

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

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

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

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

Visualizar relatórios do dispositivo

- Na seção Dispositivos, escolha o dispositivo.

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

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

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

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

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

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

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

Visualizar relatórios do conjunto de testes

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

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

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

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

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

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

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

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

Visualizar relatórios de teste

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

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

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

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

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

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

A seção Logs exibe as seguintes informações:

- Origem representa a origem de uma entrada de log. Os possíveis valores incluem:
  - Harness representa uma entrada de log que o Device Farm criou. Essas entradas de log normalmente são criadas durante os eventos de início e interrupção.

  - Dispositivo representa uma entrada de log que o dispositivo criou. Para Android, essas entradas de log são compatíveis com logcat. Para iOS, essas entradas de log são compatíveis com syslog.
  - Teste representa uma entrada de log criada por um teste ou pela estrutura de teste.
- Hora representa o tempo decorrido entre a primeira entrada de log e a entrada de log em questão. A hora é expressa em *MM:SS.SSS* formato, onde *M* representa minutos e *S* representa segundos.
- PID representa o identificador de processo (PID) que criou a entrada de log. Todas as entradas de log criadas por um aplicativo em um dispositivo têm o mesmo PID.
- Nível representa o nível de registro relativo à entrada de log. Por exemplo, `Logger.debug("This is a message!")` registra o Nível de `Debug`. Estes são os valores possíveis:
  - Alerta
  - Crítico
  - Depure
  - Emergência
  - Erro
  - Com erro
  - Com falha
  - Informações
  - Interno
  - Aviso
  - Aprovada
  - Skipped
  - Interrompido
  - Detalhado
  - Avisado
  - Aviso
- Tag representa metadados arbitrários relativos à entrada de log. Por exemplo, o logcat para Android pode usar esse recurso para descrever qual parte do sistema criou a entrada de log (por exemplo, `ActivityManager`).
- Mensagem representa a mensagem ou os dados relativos à entrada de log. Por exemplo, `Logger.debug("Hello, World!")` registra uma Mensagem de `"Hello, World!"`.

Para exibir apenas uma parte das informações:

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

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

Status dos resultados do teste do Device Farm

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

Tópicos

Status de um teste individual

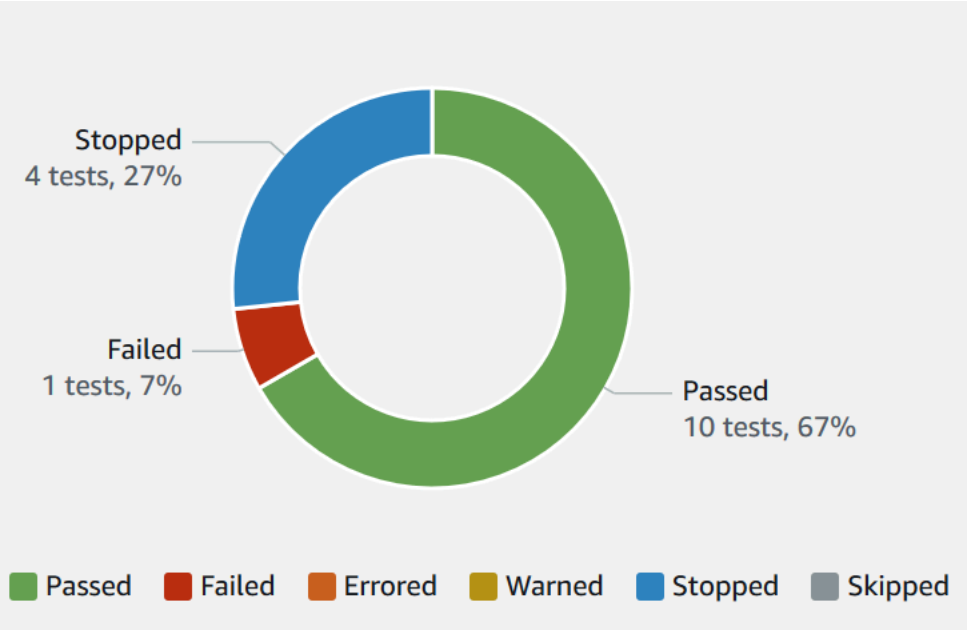
Para relatórios que descrevem um teste individual, o Device Farm exibe um ícone que representa o status do resultado do teste:

Description	Ícone
O teste bem-sucedido.	
Falha no teste.	
O Device Farm ignorou o teste.	
O teste foi interrompido.	

Description	Ícone
O Device Farm retornou um aviso.	
O Device Farm retornou um erro.	

Status de vários testes

Se você escolher uma execução concluída, o Device Farm exibirá um grafo de resumo mostrando a porcentagem de testes em vários estados.



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

Os grafos são sempre codificados por cores e rotulados.

Baixar artefatos no Device Farm

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

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

Arquivos

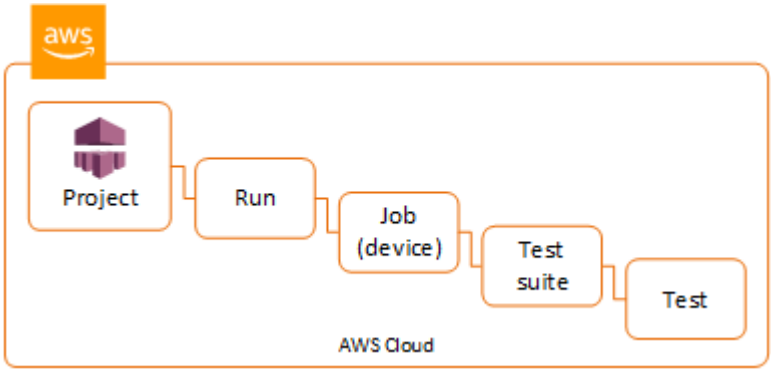
Arquivos gerados durante a execução do teste, incluindo relatórios do Device Farm. Para obter mais informações, consulte Visualizar relatórios de testes no Device Farm.

Logs

A saída de cada teste na execução.

Capturas de tela

Imagens de tela gravadas para cada teste na execução.



Baixar artefatos (console)

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

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

Baixar artefatos (AWS CLI)

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

Tópicos

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

- Etapa 3: fazer download dos artefatos

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

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

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

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

Exemplo:

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

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

```
{
    "tests": [
        {
            "status": "COMPLETED",
            "name": "Tests.FixturesTest.testExample",
            "created": 1537563725.116,
            "deviceMinutes": {
                "unmetered": 0.0,
                "total": 1.89,
```

```
                "metered": 1.89
            },
            "result": "PASSED",
            "message": "testExample passed",
            "arn": "arn:aws:devicefarm:us-west-2:123456789101:test:5e01a8c7-c861-4c0a-
b1d5-12345EXAMPLE",
            "counters": {
                "skipped": 0,
                "warned": 0,
                "failed": 0,
                "stopped": 0,
                "passed": 1,
                "errored": 0,
                "total": 1
            }
        }
    ]
}
```

Etapa 2: listar os artefatos

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

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

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

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

  A resposta contém um URL de download de cada artefato.

  ```
  {
      "artifacts": [
          {
              "url": "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/
  ExampleURL",
              "extension": "txt",
              "type": "APPIUM_JAVA_OUTPUT",
              "name": "Appium Java Output",
  ```

```
            "arn": "arn:aws:devicefarm:us-west-2:123456789101:artifact:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE",
        }
    ]
}
```

Etapa 3: fazer download dos artefatos

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

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

Baixar artefatos (API)

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

Baixar artefatos em um ambiente de teste personalizado

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

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

Saída da especificação de teste

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

Artefatos do cliente

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

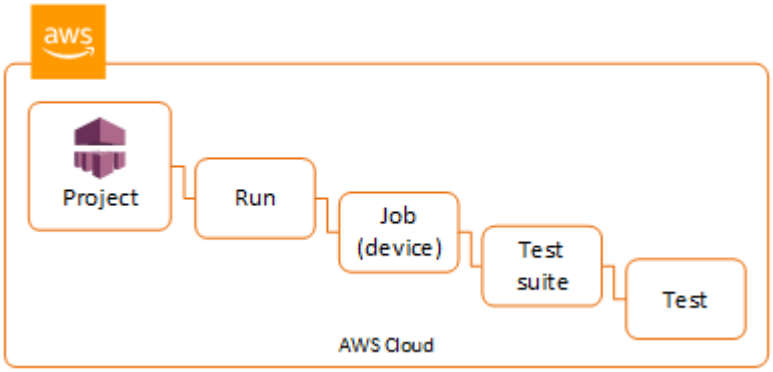
Script de shell da especificação de teste

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

Arquivo de especificação de teste

O arquivo YAML usado na execução de teste.

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



Marcação de recursos do AWS Device Farm

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

É possível usar tags para:

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

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

Tópicos

Marcar recursos

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

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

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

```
import boto3

client = boto3.client('resourcegroupstaggingapi')

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

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

Pesquisa de recursos por tag

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

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

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

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

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

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

Remover tags de recursos

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

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

Frameworks de teste e testes integrados no AWS Device Farm

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

O Device Farm executa testes automatizados fazendo com que você faça o upload do seu aplicativo e dos testes em um bucket seguro do Amazon S3 gerenciado pelo serviço. Depois de carregado, ele ativa a infraestrutura subjacente, incluindo hosts de teste gerenciados por serviços, e executa os testes paralelamente em vários dispositivos. Os resultados do teste são armazenados em um bucket S3 gerenciado pelo serviço. Essa arquitetura é chamada de execução do lado do serviço e é uma maneira rápida e eficiente de executar testes em hosts que estão fisicamente próximos ao dispositivo, sem precisar gerenciar você mesmo a infraestrutura do host de teste. Essa abordagem se adapta bem para testes em vários dispositivos de forma independente, bem como para testes a partir do contexto de um CI/CD pipeline.

Para acessar mais informações sobre testes de execução do Device Farm, consulte Ambientes de teste no AWS Device Farm.

> **Note**
>
> Para testadores do Appium, talvez você prefira executar seus testes do Appium em seu ambiente local. Com uma sessão de acesso remoto, você pode executar testes Appium do lado do cliente. Para obter mais informações, consulte o teste Appium do lado do cliente.

Testar estruturas

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

Estruturas de teste de aplicações Android

- Testes automáticos de appium
- Instrumentação

Estruturas de teste de aplicações iOS

- Testes automáticos de appium
- XCTest
- XCTest UI

Estruturas de teste de aplicações web

Aplicativos Web são compatíveis usando o Appium. Para obter mais informações sobre como trazer seus testes para o Appium, consulte Execute testes Appium automaticamente no Device Farm.

Estruturas em um ambiente de teste personalizado

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

Suporte da versão do Appium

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

Tipos de teste integrado

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

- Integrado: Fuzz (Android e iOS)

Execute testes Appium automaticamente no Device Farm

> **Note**
>
> Esta página aborda a execução de testes do Appium no ambiente de execução gerenciado do lado do servidor do Device Farm. Para executar testes do Appium em seu ambiente local do lado do cliente durante uma sessão de acesso remoto, consulte Teste do Appium do lado do cliente.

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

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

Para obter mais informações sobre testes no Device Farm e como funciona o lado do servidor, consulte. Frameworks de teste e testes integrados no AWS Device Farm

Selecionar uma versão do Appium

> **Note**
>
> O suporte para versões específicas do Appium, drivers ou programação SDKs dependerá do dispositivo e do host de teste selecionados para a execução do teste.

Os hosts de teste do Device Farm vêm pré-instalados com o Appium para permitir uma configuração mais rápida dos testes para casos de uso mais simples. No entanto, o uso do arquivo de especificação de teste permite que você instale versões diferentes do Appium, se necessário.

Cenário 1: versão pré-configurada do Appium

O Device Farm vem pré-configurado com diferentes versões do servidor Appium com base no host de teste. O host vem com ferramentas que habilitam a versão pré-configurada com o driver padrão da plataforma do dispositivo (UiAutomator2 para Android e XCUITest iOS).

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

Para ver uma lista de softwares compatíveis, consulte o tópico emSoftware compatível em ambientes de teste personalizados.

Cenário 2: versão personalizada do Appium

Para selecionar uma versão personalizada do Appium, use o `npm` comando para instalá-la. O exemplo a seguir mostra como instalar a versão mais recente do Appium 2.

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

Cenário 3: Appium em hosts iOS antigos

NoHost de teste iOS antigo, você pode escolher versões específicas do Appium com. `avm` Por exemplo, para usar o `avm` comando para definir a versão do servidor Appium como`2.1.2`, adicione esses comandos ao seu arquivo YAML de especificação de teste.

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

Seleção de uma WebDriverAgent versão para testes do iOS

Para executar testes do Appium em dispositivos iOS, WebDriverAgent é necessário o uso de. Esse aplicativo deve estar conectado para ser instalado em dispositivos iOS. O Device Farm fornece versões pré-assinadas WebDriverAgent que estão disponíveis durante a execução do ambiente de teste personalizado.

O trecho de código a seguir pode ser usado para selecionar uma WebDriverAgent versão no Device Farm dentro do seu arquivo de especificação de teste que seja compatível com a versão do seu driver de XCTest interface de usuário.

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

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

Para obter mais informações sobre o WebDriverAgent, consulte a documentação da Appium.

Integrar testes do Appium com o Device Farm

Use as instruções a seguir para integrar os testes do Appium ao AWS Device Farm. Para acessar mais informações sobre o uso de testes do Appium no Device Farm, consulte Execute testes Appium automaticamente no Device Farm.

Configurar o pacote de testes do Appium

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

Java (JUnit)

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

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

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

O plug-in a seguir cria seu código-fonte de teste (qualquer coisa no diretório `src/test`) em um arquivo JAR:

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

3. Modifique `pom.xml` para usar `maven-dependency-plugin` a fim de criar dependências como arquivos JAR.

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

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

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

O XML a seguir é uma definição de montagem que, quando configurada, instrui o Maven a criar um arquivo .zip que contém tudo o que está na raiz do diretório de saída da compilação e no diretório `dependency-jars`:

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

5. Modifique `pom.xml` para usar `maven-assembly-plugin` a fim de empacotar testes e todas as dependências em um único arquivo .zip.

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

```
<plugin>
  <artifactId>maven-assembly-plugin</artifactId>
```

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

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

Java (TestNG)

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

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

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

   O plug-in a seguir cria seu código-fonte de teste (qualquer coisa no diretório `src/test`) em um arquivo JAR:

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

3. Modifique `pom.xml` para usar `maven-dependency-plugin` a fim de criar dependências como arquivos JAR.

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

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

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

   O XML a seguir é uma definição de montagem que, quando configurada, instrui o Maven a criar um arquivo .zip que contém tudo o que está na raiz do diretório de saída da compilação e no diretório `dependency-jars`:

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

5. Modifique `pom.xml` para usar `maven-assembly-plugin` a fim de empacotar testes e todas as dependências em um único arquivo .zip.

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

```
<plugin>
```

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

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

Node.JS

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

- Gerenciador de versão do Node (nvm)

Use essa ferramenta ao desenvolver e empacotar seus testes para que dependências desnecessárias não sejam incluídas no pacote de testes.

- Node.js
- npm-bundle (instalado globalmente)

1. Verifique se o nvm está presente

   ```
   command -v nvm
   ```

   Você verá `nvm` como resultado.

   Para obter mais informações, consulte nvm on. GitHub

2. Execute este comando para instalar o Node.js:

   ```
   nvm install node
   ```

   É possível especificar uma determinada versão do Node.js:

   ```
   nvm install 11.4.0
   ```

3. Verifique se a versão correta do Node está em uso:

   ```
   node -v
   ```

4. Instale npm-bundle globalmente:

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

Python

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

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

```
$ source bin/activate
```

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

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

```
$ pip install pytest
```

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

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

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

```
$ find tests/
```

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

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

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

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

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

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

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

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

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

Ruby

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

- Gerenciador de versão do Ruby (RVM)

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

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

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

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

2. Verifique se o rvm está instalado corretamente

```
command -v rvm
```

Você verá `rvm` como resultado.

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

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

Verifique se você está na versão solicitada do Ruby:

```
ruby -v
```

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

```
bundle config specific_platform true
```

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

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

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

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

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

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

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

Criar um arquivo de pacote de teste compactado

> **Warning**
>
> No Device Farm, a estrutura de pastas dos arquivos em seu pacote de teste compactado é importante, e algumas ferramentas de arquivamento alterarão a estrutura do seu arquivo ZIP implicitamente. Recomendamos que você siga os utilitários da linha de comando

especificados abaixo, em vez de usar os utilitários de arquivamento incorporados ao gerenciador de arquivos da área de trabalho local (como o Finder ou o Windows Explorer).

Agora, empacote seus testes para o Device Farm.

Java (JUnit)

Crie e empacote seus testes:

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

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

Java (TestNG)

Crie e empacote seus testes:

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

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

Node.JS

1. Confira o projeto.

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

2. Execute este comando para instalar suas dependências locais.

   ```
   npm install
   ```

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

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

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

```
npm-bundle
```

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

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

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

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

Python

Python 2

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

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

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

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

Python 3

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

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

Ruby

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

```
# myGemset is the name of your virtual Ruby environment
```

```
rvm gemset create myGemset
```

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

```
rvm gemset use myGemset
```

3. Confira o código-fonte.

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

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

```
bundle install
```

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

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

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

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

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

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

Fazer upload do pacote de testes no Device Farm

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

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

2. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e, em seguida, Projetos.
3. Se você for um novo usuário, escolha Novo projeto, insira um nome para o projeto e escolha Enviar.

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

   Em Configurações de execução, escolha Aplicação Android se estiver testando uma aplicação Android (.apk), ou Aplicação iOS se estiver testando uma aplicação iOS (.ipa). Depois, em Selecionar aplicação, selecione Fazer upload de sua aplicação para fazer upload do pacote distribuível da aplicação.

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

   Para testes de aplicativos Web para dispositivos móveis

   Em Configurações de execução, escolha Aplicativo web.
7. Em Configurar teste, na seção Selecionar framework de teste, escolha a estrutura Appium com a qual você testa e, depois, faça upload de seu próprio pacote de teste.
8. Procure e escolha o arquivo .zip que contém os testes. O arquivo .zip deve seguir o formato descrito em Configurar o pacote de testes do Appium.
9. Siga as instruções para selecionar os dispositivos e iniciar a execução. Para obter 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 no runtime. As capturas de tela são inseridas automaticamente nos relatórios do Device Farm. Para visualizar as capturas de tela, no console do Device Farm, selecione a seção Capturas de tela.

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

Testes 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.

Para acessar mais informações sobre testes no Device Farm, consulte Frameworks de teste e testes integrados no AWS Device Farm.

Estruturas de teste de aplicações Android

Os testes personalizados estão disponíveis para dispositivos Android.

- Testes automáticos de appium
- Instrumentação

Tipos de teste incorporados para Android

Há um tipo de teste integrado disponível para dispositivos Android.

- Integrado: Fuzz (Android e iOS)

Instrumentação para Android e AWS Device Farm

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

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

Para acessar mais informações sobre testes no Device Farm, consulte Frameworks de teste e testes integrados no AWS Device Farm.

Tópicos

O que é instrumentação?

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

Considerações sobre testes de instrumentação do Android

Ao usar a instrumentação do Android, pense nas recomendações e observações a seguir.

Conferir a compatibilidade do sistema operacional Android

Confira a documentação do Android para garantir que a instrumentação seja compatível com sua versão do sistema operacional Android.

Executar pela linha de comando

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

Animações de sistema

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

Gravadores de teste

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

Análise de teste em modo padrão

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

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

Para testar isso em um ambiente local:

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

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

Integrar a instrumentação do Android ao Device Farm

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

Upload dos testes de instrumentação para Android

Use o console do Device Farm para carregar seus testes.

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

2. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e, em seguida, Projetos.
3. Na lista de projetos, escolha o projeto para o qual deseja carregar seus testes.

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

4. Selecione Criar execução.
5. Em Selecionar aplicação, na seção Opções de seleção de aplicação, escolha Fazer upload da própria aplicação.
6. Procure e escolha o arquivo de seu aplicativo Android. O arquivo deve ser .apk.
7. Na página Configurar teste, na seção Selecionar framework de teste, escolha Instrumentação e, depois, selecione Escolher arquivo.
8. Procure e escolha o arquivo .apk que contém os testes.
9. Conclua as instruções restantes para selecionar dispositivos e iniciar a execução.

(Opcional) Fazer captura de telas em testes de instrumentação do Android

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

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

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

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

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

- `/sdcard/robotium-screenshots`
- `/sdcard/test-screenshots`

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

Testes de iOS no AWS Device Farm

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

Para acessar mais informações sobre testes no Device Farm, consulte Frameworks de teste e testes integrados no AWS Device Farm.

Estruturas de teste de aplicações iOS

Os testes a seguir estão disponíveis para dispositivos iOS.

- Testes automáticos de appium
- XCTest
- XCTest UI

Tipos de teste integrados para iOS

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

- Integrado: Fuzz (Android e iOS)

Integrando o Device Farm com XCTest o iOS

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

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

Para acessar mais informações sobre testes no Device Farm, consulte Frameworks de teste e testes integrados no AWS Device Farm.

Tópicos

Crie os pacotes para sua XCTest corrida

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

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

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

Para gerar a saída da compilação para seu aplicativo

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

Para criar o pacote de aplicativos

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

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

6. Opcionalmente, você pode excluir a pasta `Payload` e o arquivo `.app` que está nela.

Para criar o XCTest pacote

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

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

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

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

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

   Caso contrário, no painel de navegação do Device Farm, escolha Teste para dispositivos móveis e escolha Projetos.
3. Escolha o projeto que você deseja usar para executar o teste.
4. Escolha Criar execução.
5. Em Configurações de execução, na seção Tipo de execução, escolha Aplicação iOS.
6. Em Selecionar aplicação, na seção Opções de seleção de aplicação, escolha Fazer upload da própria aplicação. Depois, selecione Escolher arquivo em Fazer upload de aplicação.
7. Procure o arquivo `.ipa` para seu aplicativo e faça o upload dele.

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

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

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

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

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

Para acessar informações gerais sobre testes no Device Farm, consulte Frameworks de teste e testes integrados no AWS Device Farm.

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

Tópicos

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

Prepare seus testes de XCTest interface do iOS

Você pode fazer upload de um arquivo `.ipa` ou `.zip` para seu pacote de testes XCTEST_UI.

Arquivo `.ipa` é um arquivo de aplicação que contém a aplicação iOS Runner em formato de pacote. Arquivos adicionais não podem ser incluídos no arquivo *`.ipa`*.

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

Opção 1: criar um pacote XCTest UI .ipa

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

Para criar um arquivo .ipa:

1. Crie um diretório denominado *`Payload`*.
2. Adicione o diretório da sua aplicação ao diretório da carga útil.
3. Arquive o diretório Carga útil em um arquivo `.zip` e mude a extensão do arquivo para `.ipa`.

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

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

Opção 2: criar um pacote XCTest UI .zip

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

```
.
### swift-sample-UI.zip (directory)
   ### my-project-nameUITest-Runner.app [OR] my-project-nameUITest.ipa
```

```
   ### SampleTestPlan_2.xctestrun
   ### SampleTestPlan_1.xctestrun
   ### (any other files)
```

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

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

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

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

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

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

Carregue seus testes de XCTest interface do iOS

Use o console do Device Farm para carregar seus testes.

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

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

4. Escolha Criar execução.
5. Em Configurações de execução, na seção Tipo de execução, escolha Aplicação iOS.
6. Em Selecionar aplicação, na seção Opções de seleção de aplicação, escolha Fazer upload da própria aplicação. Depois, selecione Escolher arquivo em Fazer upload de aplicação.
7. Procure e escolha o arquivo de seu aplicativo iOS. O arquivo deve ser .ipa.

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

8. Em Configurar teste, na seção Selecionar estrutura de teste, escolha XCTest UI. Depois, selecione Escolher arquivo em Fazer upload de aplicação.
9. Procure e escolha o arquivo.ipa ou.zip que contém seu executor de teste de XCTest interface do iOS.
10. Conclua as etapas restantes no processo de criação do projeto. Você selecionará os dispositivos em que deseja testar e, opcionalmente, especificar uma configuração adicional.
11. Escolha Criar execução. O Device Farm executa o teste e mostra os resultados no console.

Testes de aplicações web no AWS Device Farm

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

Para acessar mais informações sobre testes no Device Farm, consulte Frameworks de teste e testes integrados no AWS Device Farm.

Regras para dispositivos de acesso limitado e ilimitado

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

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

Veja outro exemplo: suponhamos que você tenha cinco dispositivos Android de acesso ilimitado e 0 dispositivo iOS de acesso ilimitado. Se selecionar apenas dispositivos Android para a execução web, seus dispositivos de acesso ilimitado serão usados. Se selecionar dispositivos Android e iOS para a execução web, o método de cobrança será monitorado e seus dispositivos de acesso ilimitado não serão usados.

Testes integrados no AWS Device Farm

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

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

- Integrado: Fuzz (Android e iOS): o teste fuzz envia aleatoriamente eventos de interface de usuário aos dispositivos e, depois relata os resultados.

Para acessar mais informações sobre testes e frameworks de testes no Device Farm, consulte Frameworks de teste e testes integrados no AWS Device Farm.

Executar o teste fuzz incorporado do Device Farm (Android e iOS)

O teste integrado fuzz do Device Farm envia aleatoriamente eventos de interface do usuário aos dispositivos e, depois, relata os resultados.

Para acessar mais informações sobre testes no Device Farm, consulte Frameworks de teste e testes integrados no AWS Device Farm.

Como executar o teste 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 Teste para dispositivos móveis e, em seguida, Projetos.
3. Na lista de projetos, escolha o projeto em que você deseja executar o teste 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. Escolha Criar execução.
5. Em Configurações de execução, selecione o tipo de execução na seção Tipo de execução. Selecione Aplicação Android se você não tiver uma aplicação pronta para teste ou se estiver testando uma aplicação Android (.apk). Selecione Aplicação iOS se estiver testando uma aplicação iOS (.ipa).
6. Em Selecionar aplicação, escolha Selecionar aplicação de exemplo fornecida pelo Device Farm se você não tiver uma aplicação disponível para teste. Se você estiver trazendo sua própria aplicação, selecione Fazer upload da própria aplicação e escolha o arquivo da sua aplicação.
7. Em Configurar teste, na seção Selecionar framework de teste, escolha Incorporado: fuzz.
8. Se qualquer uma das configurações a seguir forem exibidas, você poderá aceitar os valores padrão ou especificar seu próprio:

   - Contagem de eventos: especifique um número entre 1 e 10.000, que representa o número de eventos de interface de usuário que o teste Fuzz deve executar.

- Restrição de eventos: especifique um número entre 0 e 1.000, que representa o número de milissegundos que o teste de fuzz deve aguardar para realizar o próximo evento de interface de usuário.
- Propagação aleatória: especifique um número para o teste de fuzz usar para randomizar eventos de interface de usuário. A especificação de um mesmo número de testes Fuzz subsequentes garante sequências de eventos idênticas.

9. Conclua as instruções restantes para selecionar dispositivos e iniciar a execução.

Ambiente de teste personalizado no AWS Device Farm.

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

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

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

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

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

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

Tópicos

- Variáveis de ambiente para ambientes de teste personalizados
- Práticas recomendadas para execução de ambientes de teste personalizados
- Migrar testes de um ambiente padrão para um ambiente de teste personalizado
- Extensão de ambientes de teste personalizados no Device Farm

Referência e sintaxe da especificação de teste

A especificação de teste (especificação de teste) é um arquivo que você usa para definir ambientes de teste personalizados no Device Farm.

Fluxo de trabalho de especificações de teste

A especificação de teste do Device Farm executa as fases e seus comandos em uma ordem predeterminada, permitindo que você personalize a forma como seu ambiente é preparado e executado. Quando cada fase é executada, seus comandos são executados na ordem listada no arquivo de especificação de teste. As fases são executadas na seguinte sequência

1. `install`- É aqui que ações como baixar, instalar e configurar ferramentas devem ser definidas.
2. `pre_test`- É aqui que as ações de pré-teste, como iniciar processos em segundo plano, devem ser definidas.
3. `test`- É aqui que o comando que invoca seu teste deve ser definido.
4. `post_test`- É aqui que todas as tarefas finais que precisam ser executadas após a conclusão do teste devem ser definidas, como geração de relatórios de teste e agregação de arquivos de artefatos.

Sintaxe da especificação de teste

A seguir está o esquema YAML para um arquivo de especificação de teste

```
version: 0.1

android_test_host: "string"
ios_test_host: "string"

phases:
  install:
```

```
    commands:
      - "string"
      - "string"
  pre_test:
    commands:
      - "string"
      - "string"
  test:
    commands:
      - "string"
      - "string"
  post_test:
    commands:
      - "string"
      - "string"

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

`version`

(Obrigatório, número)

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

`android_test_host`

(Opcional, string)

O host de teste que será selecionado para execuções de teste realizadas em dispositivos Android. Esse campo é obrigatório para testes em dispositivos Android. Para obter mais informações, consulte Hosts de teste disponíveis para ambientes de teste personalizados.

`ios_test_host`

(Opcional, string)

O host de teste que será selecionado para execuções de teste realizadas em dispositivos iOS. Esse campo é obrigatório para execuções de teste em dispositivos iOS com uma versão principal maior que 26. Para obter mais informações, consulte Hosts de teste disponíveis para ambientes de teste personalizados.

`phases`

Esta seção contém grupos de comandos executados durante uma execução de teste, em que cada fase é opcional. Os nomes de fases de teste permitidos são: `install` `pre_testtest`,, `post_test` e.

- `install`- As dependências padrão para estruturas de teste suportadas pelo Device Farm já estão instaladas. Essa fase contém comandos adicionais, se houver, que o Device Farm executa durante a instalação.
- `pre_test`- Os comandos, se houver, executados antes do teste automatizado.
- `test`- Os comandos executados durante a execução do teste automatizado. Se algum comando na fase de teste falhar (o que significa que ele retorna um código de saída diferente de zero), o teste é marcado como falhado
- `post_test`- Os comandos, se houver, executados após a execução do teste automatizado. Isso será executado independentemente de seu teste na `test` fase ser bem-sucedido ou falhar.

`commands`

(Opcional, Listar [string])

Uma lista de strings a serem executadas como um comando shell durante a fase.

`artifacts`

(Opcional, Listar [string])

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

Esses artefatos de teste estão disponíveis para cada dispositivo na execução de teste. Para obter informações sobre como recuperar os artefatos de teste, consulte Baixar 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.

Exemplos de especificações de teste

Os exemplos a seguir mostram especificações de teste que podem ser executadas no Device Farm.

Simple Demo

Veja a seguir um exemplo de arquivo de especificação de teste que simplesmente é registrado `Hello world!` como um artefato de execução de teste.

```
version: 0.1

android_test_host: amazon_linux_2
ios_test_host: macos_sequoia

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

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

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

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

artifacts:
```

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

Appium Android

A seguir está um exemplo de arquivo de especificação de teste que configura um teste Appium Java TestNG executado no Android.

```
version: 0.1

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

phases:

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

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

      # The Device Farm service periodically updates the preinstalled Appium
 versions over time to
      # incorporate the latest minor and patch versions for each major version. If
 you wish to
```

```
    # select a specific version of Appium, you can use NPM to install it.
    # - npm install -g appium@2.19.0

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

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

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

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

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

```
      # https://docs.aws.amazon.com/devicefarm/latest/developerguide/custom-test-
environment-variables.html
      - |-
        appium --base-path=$APPIUM_BASE_PATH --log-timestamp \
          --log-no-colors --relaxed-security --default-capabilities \
          "{\"appium:deviceName\": \"$DEVICEFARM_DEVICE_NAME\", \
          \"platformName\": \"$DEVICEFARM_DEVICE_PLATFORM_NAME\", \
          \"appium:udid\":\"$DEVICEFARM_DEVICE_UDID\", \
          \"appium:platformVersion\": \"$DEVICEFARM_DEVICE_OS_VERSION\", \
          \"appium:chromedriverExecutableDir\":
 \"$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR\", \
          \"appium:automationName\": \"UiAutomator2\"}" \
          >> $DEVICEFARM_LOG_DIR/appium.log 2>&1 &;

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

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

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

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

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

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

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

Appium iOS

A seguir está um exemplo de arquivo de especificação de teste que configura um teste Appium Java TestNG executado no iOS.

```
version: 0.1

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

```
phases:

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

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

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

      # When running iOS tests with Appium version 2, the XCUITest driver is
 preinstalled using driver
      # version 9.10.5 for Appium 2.5.4. If you want to install a different version
 of the driver,
      # you can use the Appium extension CLI to uninstall the existing XCUITest
 driver
      # and install your desired version:
      # - |-
      #   if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "iOS" ];
      #   then
      #     appium driver uninstall xcuitest;
      #     appium driver install xcuitest@10.0.0;
```

```
     #   fi;

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

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

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

     # Device Farm provides multiple pre-built versions of WebDriverAgent (WDA), a
required
     # Appium dependency for iOS, where each version corresponds to the XCUITest
driver version selected.
     # If Device Farm cannot find a corresponding version of WDA for your XCUITest
driver,
     # the latest available version is selected by default.
     - |-
       APPIUM_DRIVER_VERSION=$(appium driver list --installed --json | jq -r
".xcuitest.version" | cut -d "." -f 1);
       CORRESPONDING_APPIUM_WDA=$(env | grep
"DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V${APPIUM_DRIVER_VERSION}")
       if [[ ! -z "$APPIUM_DRIVER_VERSION" ]] && [[ ! -z
"$CORRESPONDING_APPIUM_WDA" ]]; then
         echo "Using Device Farm's prebuilt WDA version ${APPIUM_DRIVER_VERSION}.x,
which corresponds with your driver";
         DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH=$(echo $CORRESPONDING_APPIUM_WDA |
cut -d "=" -f2)
       else
         LATEST_SUPPORTED_WDA_VERSION=$(env | grep
"DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V" | sort -V -r | head -n 1)
         echo "Unknown driver version $APPIUM_DRIVER_VERSION; falling back to the
Device Farm default version of $LATEST_SUPPORTED_WDA_VERSION";
```

```
          DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH=$(echo
 $LATEST_SUPPORTED_WDA_VERSION | cut -d "=" -f2)
        fi;

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

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

      # This code snippet is to wait until the Appium server starts.
      - |-
        appium_initialization_time=0;
```

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

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

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

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

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

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

```
  post_test:
    commands:

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

Appium (Both Platforms)

A seguir está um exemplo de arquivo de especificação de teste que configura um teste Appium Java TestNG executado no Android e no iOS.

```
version: 0.1

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

phases:

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

```
    # you first need to set up a Node.js environment that is compatible with your
version of Appium.
    - devicefarm-cli use node 20
    - node --version

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

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

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

    # When running iOS tests with Appium version 2, the XCUITest driver is
preinstalled using driver
    # version 9.10.5 for Appium 2.5.4. If you want to install a different version
of the driver,
    # you can use the Appium extension CLI to uninstall the existing XCUITest
driver
    # and install your desired version:
    # - |-
    #   if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "iOS" ];
    #   then
    #     appium driver uninstall xcuitest;
    #     appium driver install xcuitest@10.0.0;
    #   fi;
```

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

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

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

     # Device Farm provides multiple pre-built versions of WebDriverAgent (WDA), a
required
     # Appium dependency for iOS, where each version corresponds to the XCUITest
driver version selected.
     # If Device Farm cannot find a corresponding version of WDA for your XCUITest
driver,
     # the latest available version is selected by default.
     - |-
       if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "iOS" ]; then
         APPIUM_DRIVER_VERSION=$(appium driver list --installed --json | jq -r
".xcuitest.version" | cut -d "." -f 1);
         CORRESPONDING_APPIUM_WDA=$(env | grep
"DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V${APPIUM_DRIVER_VERSION}")
         if [[ ! -z "$APPIUM_DRIVER_VERSION" ]] && [[ ! -z
"$CORRESPONDING_APPIUM_WDA" ]]; then
           echo "Using Device Farm's prebuilt WDA version
${APPIUM_DRIVER_VERSION}.x, which corresponds with your driver";
           DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH=$(echo $CORRESPONDING_APPIUM_WDA
| cut -d "=" -f2)
         else
           LATEST_SUPPORTED_WDA_VERSION=$(env | grep
"DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V" | sort -V -r | head -n 1)
           echo "Unknown driver version $APPIUM_DRIVER_VERSION; falling back to the
Device Farm default version of $LATEST_SUPPORTED_WDA_VERSION";
```

```
            DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH=$(echo
 $LATEST_SUPPORTED_WDA_VERSION | cut -d "=" -f2)
          fi;
        fi;

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

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

```
            \"platformName\": \"$DEVICEFARM_DEVICE_PLATFORM_NAME\", \
            \"appium:udid\":\"$DEVICEFARM_DEVICE_UDID_FOR_APPIUM\", \
            \"appium:platformVersion\": \"$DEVICEFARM_DEVICE_OS_VERSION\", \
            \"appium:derivedDataPath\": \"$DEVICEFARM_WDA_DERIVED_DATA_PATH\", \
            \"appium:usePrebuiltWDA\": true, \
            \"appium:automationName\": \"XCUITest\"}" \
            >> $DEVICEFARM_LOG_DIR/appium.log 2>&1 &
        fi;

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

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

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

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

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

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

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

Hosts para ambientes de teste personalizados

O Device Farm oferece suporte a um conjunto de sistemas operacionais com software pré-configurado por meio do uso de um ambiente de host de teste. Durante a execução do teste, o Device Farm utiliza instâncias gerenciadas pela Amazon (hosts) que se conectam dinamicamente ao dispositivo selecionado em teste. Essa instância é totalmente limpa e não é reutilizada entre as execuções e é encerrada com os artefatos gerados após a conclusão da execução do teste.

Tópicos

- Ambiente de teste para dispositivos Android
- Ambiente de teste para dispositivos iOS

Hosts de teste disponíveis para ambientes de teste personalizados

Os hosts de teste são totalmente gerenciados pelo Device Farm. A tabela a seguir lista os hosts de teste Device Farm atualmente disponíveis e compatíveis para ambientes de teste personalizados.

Plataforma de dispositivos	Host de teste	Sistema operacional	Arquitetura (s)	Dispositivos compatíveis
Android	amazon_linux_2	Amazon Linux 2	x86_64	Android6 e acima
iOS	macos_sequoia	macOS Sequoia(versão 15)	arm64	iOS15 a 26

> **Note**
>
> Periodicamente, o Device Farm adiciona novos hosts de teste para uma plataforma de dispositivo para oferecer suporte às versões mais recentes do sistema operacional do dispositivo e suas dependências. Quando isso ocorre, os hosts de teste mais antigos da respectiva plataforma do dispositivo estão sujeitos ao fim do suporte.

Versão do sistema operacional

Cada host de teste disponível usa uma versão específica do sistema operacional compatível com o Device Farm no momento. Embora tentemos usar a versão mais recente do sistema operacional, essa pode não ser a versão mais recente distribuída publicamente disponível. O Device Farm atualizará periodicamente o sistema operacional com pequenas atualizações de versão e patches de segurança.

Para saber a versão específica (incluindo a versão secundária) do sistema operacional em uso durante a execução do teste, você pode adicionar o seguinte trecho de código a qualquer uma das fases do arquivo de especificação de teste.

Example

```
phases:
  install:
    commands:
      # The following example prints the instance's operating system version details
      - |-
        if [[ "Darwin" == "$(uname)" ]]; then
          echo "$(sw_vers --productName) $(sw_vers --productVersion) ($(sw_vers --
buildVersion))";
        else
          echo "$(. /etc/os-release && echo $PRETTY_NAME) ($(uname -r))";
        fi
```

Seleção de um host de teste para ambientes de teste personalizados

Você pode especificar o host de teste do Android e do iOS nas `ios_test_host` variáveis apropriadas `android_test_host` do seu arquivo de especificação de teste.

Se você não especificar uma seleção de host de teste para a plataforma de dispositivo especificada, os testes serão executados no host de teste que o Device Farm definiu como padrão para o dispositivo e a configuração de teste especificados.

> **Important**
>
> Ao testar no iOS 18 e versões anteriores, um host de teste legado será usado quando um host não for selecionado. Para obter mais informações, consulte o tópico noHost de teste iOS antigo.

Como exemplo, analise o seguinte trecho de código:

Example

```
version: 0.1
android_test_host: amazon_linux_2
ios_test_host: macos_sequoia

phases:
  # ...
```

Software compatível em ambientes de teste personalizados

O Device Farm usa máquinas host pré-instaladas com muitas das bibliotecas de software necessárias para executar estruturas de teste suportadas em nosso serviço, fornecendo um ambiente de teste pronto para o lançamento. O Device Farm oferece suporte a vários idiomas por meio do uso de nosso mecanismo de seleção de software e atualizará periodicamente as versões dos idiomas incluídos no ambiente.

Para qualquer outro software necessário, você pode modificar o arquivo de especificação de teste para instalar pelo pacote de testes, fazer o download da Internet ou acessar origens privadas na VPC (consulte VPC ENI para obter mais informações). Para obter mais informações, consulte Exemplos de especificações de teste.

Software pré-configurado

Para facilitar o teste de dispositivos em cada plataforma, as seguintes ferramentas são fornecidas no host de teste:

Ferramentas	Plataforma (s) do dispositivo
Android SDK Build-Tools	Android
Android SDK Platform-Tools(incluiadb)	Android
Xcode	iOS

Software selecionável

Além do software pré-configurado no host, o Device Farm oferece uma maneira de selecionar determinadas versões do software compatível por meio das `devicefarm-cli` ferramentas.

A tabela a seguir contém o software selecionável e os hosts de teste que o contêm.

Software/Ferramenta	Hosts que oferecem suporte a este software	Comando a ser usado em sua especificação de teste
Java 17	amazon_linux_2 macos_sequoia	`devicefarm-cli use java 17`

Software/Ferramenta	Hosts que oferecem suporte a este software	Comando a ser usado em sua especificação de teste
Java 11	amazon_linux_2 macos_sequoia	`devicefarm-cli use java 11`
Java 8	amazon_linux_2 macos_sequoia	`devicefarm-cli use java 8`
Node.js 20	amazon_linux_2 macos_sequoia	`devicefarm-cli use node 20`
Node.js 18	amazon_linux_2 macos_sequoia	`devicefarm-cli use node 18`
Node.js 16	amazon_linux_2	`devicefarm-cli use node 16`
Python 3.11	amazon_linux_2 macos_sequoia	`devicefarm-cli use python 3.11`
Python 3.10	amazon_linux_2 macos_sequoia	`devicefarm-cli use python 3.10`
Python 3.9	amazon_linux_2 macos_sequoia	`devicefarm-cli use python 3.9`
Python 3.8	amazon_linux_2	`devicefarm-cli use python 3.8`
Ruby 3.2	amazon_linux_2 macos_sequoia	`devicefarm-cli use ruby 3.2`

Software/Ferramenta	Hosts que oferecem suporte a este software	Comando a ser usado em sua especificação de teste
Ruby 2.7	amazon_linux_2	`devicefarm-cli use ruby 2.7`
Appium 3	amazon_linux_2	`devicefarm-cli use appium 3`
Appium 2	amazon_linux_2 macos_sequoia	`devicefarm-cli use appium 2`
Appium 1	amazon_linux_2	`devicefarm-cli use appium 1`
Xcode 26	macos_sequoia	`devicefarm-cli use xcode 26`
Xcode 16	macos_sequoia	`devicefarm-cli use xcode 16`

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

Usando a ferramenta devicefarm-cli em ambientes de teste personalizados

O host de teste usa uma ferramenta padronizada de gerenciamento de versões chamada `devicefarm-cli` para selecionar versões de software. Essa ferramenta é separada da AWS CLI e está disponível somente no host de teste do Device Farm. Com a `devicefarm-cli`, você pode alternar para qualquer versão de software pré-instalada no host de teste. Isso proporciona uma maneira direta de manter o arquivo de especificações de teste do Device Farm ao longo do tempo e oferece um mecanismo previsível para atualizar as versões do software no futuro.

> **Important**
>
> Essa ferramenta de linha de comando não está disponível em hosts iOS antigos. Para obter mais informações, consulte o tópico noHost de teste iOS antigo.

O trecho abaixo mostra a página de `help` da `devicefarm-cli`:

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

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

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

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

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

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

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

Para obter mais informações sobre o software pré-instalado no host de teste, consulte Software compatível em ambientes de teste personalizados.

Ambiente de teste para dispositivos 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 Amazon Linux 2 oferece várias vantagens:

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

Tópicos

- Intervalos de IP aceitos para o ambiente de teste do Amazon Linux 2 no Device Farm

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

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

Intervalos de IP
44.236.137.143
52.13.151.244
52.35.189.191
54.201.250.26

Para acessar mais informações sobre ambientes de teste no Device Farm, consulte Ambiente de teste para dispositivos Android.

Ambiente de teste para dispositivos iOS

O Device Farm utiliza instâncias macOS gerenciadas pela Amazon (hosts) que se conectam dinamicamente ao dispositivo iOS durante a execução do teste. Cada host é pré-configurado com um software que permite testar dispositivos em várias plataformas de teste populares, como XCTest UI e Appium.

A iteração atual do host de teste do iOS melhorou a experiência de teste em comparação com as versões anteriores, incluindo:

- Experiência consistente de sistema operacional e ferramentas do iOS 15 ao iOS 26 Antes, o host de teste era determinado pelo dispositivo em uso, resultando em um ambiente de software fragmentado ao ser executado em várias versões do iOS. A experiência atual permite uma seleção simples de hosts para permitir um ambiente consistente em todos os dispositivos. Isso permitirá que a mesma versão e ferramentas do macOS (como o Xcode) estejam disponíveis em cada dispositivo iOS.

- Melhorias no desempenho dos testes do iOS 15 e 16 Usando a infraestrutura atualizada, o tempo de configuração melhorou substancialmente para os testes do iOS 15 e 16.
- Versões padronizadas de software selecionáveis para dependências suportadas Agora temos o sistema de seleção de `devicefarm-cli` software nos hosts de teste iOS e Android, permitindo que você selecione sua versão preferida de nossas dependências suportadas. Para dependências compatíveis (como Java, Python, Node.js, Ruby e Appium), as versões poderão ser selecionadas por meio da especificação de teste. Para ter uma ideia de como esse recurso funciona, consulte o tópico em[Software compatível em ambientes de teste personalizados](#).

> **Important**
>
> Se estiver executando no iOS 18 e versões anteriores, seus testes serão executados em hosts de teste legados por padrão. Veja o tópico abaixo sobre como migrar dos hosts legados.

Host de teste iOS antigo

Para testes existentes no iOS 18 e versões anteriores, os hosts de teste legados são selecionados por padrão para ambientes de teste personalizados. A tabela a seguir contém a versão do host de teste que é executada pela versão do dispositivo iOS.

Sistema operacional	Arquitetura (s)	Padrão para dispositivos
macOS Sonoma(versão 14)	arm64	iOS 18
macOS Ventura(versão 13)	arm64	iOS 17
macOS Monterey(versão 12)	x86_64	iOS 16e abaixo

Para selecionar os hosts de teste mais novos, consulte o tópico relacionado[Migrando seus ambientes de teste personalizados para os novos hosts de teste do iOS](#).

Software compatível para dispositivos iOS

Para oferecer suporte aos testes de dispositivos iOS, os hosts de teste do Device Farm para dispositivos iOS vêm pré-configurados com o Xcode e suas ferramentas de linha de comando

associadas. Para outros softwares disponíveis, consulte o tópico relacionadoSoftware compatível em ambientes de teste personalizados.

Migrando seus ambientes de teste personalizados para os novos hosts de teste do iOS

Para migrar os testes existentes do host legado para o novo host de teste do macOS, você precisará desenvolver novos arquivos de especificação de teste com base nos arquivos preexistentes.

A abordagem recomendada é começar com o exemplo de arquivo de especificação de teste para os tipos de teste desejados e, em seguida, migrar os comandos relevantes do arquivo de especificação de teste antigo para o novo. Isso permite que você aproveite novos recursos e otimizações da especificação de teste de exemplo para o novo host enquanto reutiliza trechos do código existente.

Tópicos

Tutorial: migrando arquivos de especificações de teste do iOS com o console

Neste exemplo, o console Device Farm será usado para integrar uma especificação de teste de dispositivo iOS existente para usar o novo host de teste.

Etapa 1: Criar novos arquivos de especificação de teste com o console

1. Faça login no console do AWS Device Farm.
2. Navegue até o projeto do Device Farm que contém seus testes de automação.
3. Faça o download de uma cópia da especificação de teste existente com a qual você deseja integrar.
    a. Clique na opção “Configurações do projeto” e navegue até a guia Uploads.
    b. Navegue até o arquivo de especificação de teste com o qual você deseja integrar.
    c. Clique no botão Download para fazer uma cópia local desse arquivo.
4. Navegue de volta para a página do projeto e clique em Criar execução.
5. Preencha as opções no assistente como se fosse iniciar uma nova execução, mas pare na opção Selecionar especificação de teste.

6. Usando a especificação de teste do iOS selecionada por padrão, clique no botão Criar uma especificação de teste.
7. Modifique a especificação do teste que foi selecionada por padrão no editor de texto.

    a. Se ainda não estiver presente, modifique o arquivo de especificação de teste para selecionar o novo host usando:

    ```
    ios_test_host: macos_sequoia
    ```

    b. A partir da cópia de sua especificação de teste baixada em uma etapa anterior, revise cada `phase` uma.

    c. Copie comandos das fases da especificação de teste antiga para cada fase respectiva na nova especificação de teste, ignorando os comandos relacionados à instalação ou seleção de Java, Python, Node.js, Ruby, Appium ou Xcode.
8. Insira um novo nome de arquivo na caixa de texto Salvar como.
9. Clique no botão Salvar como novo para salvar suas alterações.

Para ver um exemplo de um arquivo de especificação de teste que você pode usar como referência, consulte o exemplo fornecido emExemplos de especificações de teste.

Etapa 2: Selecionar software pré-instalado

No novo host de teste, as versões pré-instaladas do software são selecionadas usando uma nova ferramenta padronizada de gerenciamento de versões chamada. `devicefarm-cli` Agora, esse conjunto de ferramentas é a abordagem recomendada para usar os vários softwares que fornecemos nos hosts de teste.

Como exemplo, você adicionaria a seguinte linha para usar um JDK 17 diferente em seu ambiente de teste:

```
- devicefarm-cli use java 17
```

Para obter mais informações sobre o software suportado disponível, revise:Software compatível em ambientes de teste personalizados.

Etapa 3: Usando o Appium e suas dependências por meio das ferramentas de seleção de software

O novo host de teste suporta apenas o Appium 2.x e superior. Selecione explicitamente a versão do Appium usando o`devicefarm-cli`, enquanto remove ferramentas legadas, como. `avm` Por exemplo:

```
# This line using 'avm' should be removed
# - avm 2.3.1

# And the following lines should be added
- devicefarm-cli use appium 2 # Selects the version
- appium --version            # Prints the version
```

A versão do Appium selecionada `devicefarm-cli` vem pré-instalada com uma versão compatível do driver XCUITest para iOS.

Além disso, você precisará atualizar sua especificação de teste para usar `DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V9` em vez de `DEVICEFARM_WDA_DERIVED_DATA_PATH`. A nova variável de ambiente aponta para uma versão pré-construída da WebDriverAgent 9.x, que é a versão mais recente compatível com os testes do Appium 2.

Para obter mais informações, revise Seleção de uma WebDriverAgent versão para testes do iOS Variáveis de ambiente para testes Appium e.

Diferenças entre os hosts de teste novos e os antigos

Ao editar seu arquivo de especificação de teste para usar o novo host de teste do iOS e fazer a transição dos testes do host de teste legado, esteja ciente dessas principais diferenças de ambiente:

- Versões do Xcode: no ambiente antigo do host de teste, a versão do Xcode disponível foi baseada na versão iOS do dispositivo usado para testes. Por exemplo, testes em dispositivos iOS 18 usaram o Xcode 16 no host legado, enquanto os testes no iOS 17 usaram o Xcode 15. No novo ambiente host, todos os dispositivos podem acessar as mesmas versões do Xcode, permitindo um ambiente consistente para testes em dispositivos com versões diferentes. Para obter uma lista das versões do Xcode atualmente disponíveis, consulteSoftware compatível.
- 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 novo host de teste 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, `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, ao usar a `devicefarm-cli` para selecionar suas versões de software, você deve selecionar as versões que estão na lista de versões compatíveis.
- O **`libimobiledevice`** conjunto de ferramentas foi removido em favor de ferramentas mais novas/originais para monitorar os testes atuais de dispositivos iOS e os padrões do setor. Para o iOS 17 e versões posteriores, você pode migrar a maioria dos comandos para usar ferramentas semelhantes do Xcode, chamadas. `devicectl` Para obter informações sobre`devicectl`, você pode executar `xcrun devicectl help` a partir de uma máquina com o Xcode instalado.
- Os caminhos de arquivo codificados no arquivo de especificação de teste do host legado como caminhos absolutos provavelmente não funcionarão conforme o esperado no novo host de teste e geralmente não são recomendados para o uso do arquivo de especificação de teste. Recomendamos que você use caminhos relativos e variáveis de ambiente para todo o código do arquivo de especificação de teste. Para obter mais informações, consulte o tópico emPráticas recomendadas para execução de ambientes de teste personalizados.
- Versão e arquitetura do sistema operacional: os hosts de teste antigos estavam usando uma variedade de versões do macOS e arquiteturas de CPU com base no dispositivo atribuído. Como resultado, os usuários podem notar algumas diferenças nas bibliotecas de sistema disponíveis no ambiente. Para obter mais informações sobre a versão anterior do sistema operacional host, reviseHost de teste iOS antigo.
- Para usuários do Appium, a forma de selecionar o WebDriverAgent mudou para um prefixo de variável de ambiente de uso `DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V` em vez do prefixo antigo. `DEVICEFARM_WDA_DERIVED_DATA_PATH_V` Para obter mais informações sobre a variável atualizada, reviseVariáveis de ambiente para testes Appium.
- 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.

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.

Acesse os recursos da AWS usando uma função de execução do IAM

O Device Farm suporta a especificação de uma função do IAM que será assumida pelo ambiente de execução de teste personalizado durante a execução do teste. Esse recurso permite que seus testes acessem com segurança os recursos da AWS em sua conta, como buckets do Amazon S3, tabelas do DynamoDB ou outros serviços da AWS dos quais seu aplicativo depende.

Tópicos

Visão geral do

Quando você especifica uma função de execução do IAM, o Device Farm assume essa função durante a execução do teste, permitindo que seus testes interajam com os serviços da AWS usando as permissões definidas na função.

Os casos de uso comuns para funções de execução do IAM incluem:

- Acessando dados de teste armazenados em buckets do Amazon S3
- Enviando artefatos de teste para os buckets do Amazon S3
- Recuperando a configuração do aplicativo da AWS AppConfig
- Escrevendo registros de teste e métricas para a Amazon CloudWatch
- Envio de resultados de testes ou mensagens de status para filas do Amazon SQS
- Chamando funções do AWS Lambda como parte dos fluxos de trabalho de teste

Requisitos de função do IAM

Para usar uma função de execução do IAM com o Device Farm, sua função deve atender aos seguintes requisitos:

- Relação de confiança: o diretor do serviço Device Farm deve ser confiável para assumir a função. A política de confiança deve ser incluída `devicefarm.amazonaws.com` como uma entidade confiável.
- Permissões: a função deve ter as permissões necessárias para acessar os recursos da AWS que seus testes exigem.
- Duração da sessão: a duração máxima da sessão da função deve ser pelo menos tão longa quanto a configuração de tempo limite de trabalho do seu projeto Device Farm. Por padrão, os projetos do Device Farm têm um tempo limite de trabalho de 150 minutos, portanto, sua função deve suportar uma duração de sessão de pelo menos 150 minutos.
- Mesmo requisito de conta: a função do IAM deve estar na mesma conta da AWS usada para chamar o Device Farm. A suposição de função entre contas não é suportada.
- PassRole permissão: o chamador deve ser autorizado a transmitir a função do IAM por meio de uma política que permita a `iam:PassRole` ação na função de execução especificada.

Exemplo de política de confiança

O exemplo a seguir mostra uma política de confiança que permite que o Device Farm assuma sua função de execução. Essa política de confiança só deve ser anexada à função específica do IAM que você pretende usar com o Device Farm, não a outras funções em sua conta:

Example

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "devicefarm.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

Exemplo de política de permissões

O exemplo a seguir mostra uma política de permissões que concede acesso a serviços comuns da AWS usados em testes:

Example

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::my-test-bucket",
        "arn:aws:s3:::my-test-bucket/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "appconfig:GetConfiguration",
        "appconfig:StartConfigurationSession"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:*:*:log-group:/devicefarm/test-*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "sqs:SendMessage",
        "sqs:GetQueueUrl"
      ],
      "Resource": "arn:aws:sqs:*:*:test-results-*"
    }
  ]
}
```

Configurando uma função de execução do IAM

Você pode especificar uma função de execução do IAM no nível do projeto ou para execuções de teste individuais. Quando configuradas no nível do projeto, todas as execuções dentro desse projeto herdarão a função de execução. Uma função de execução configurada em uma execução substituirá qualquer função configurada em seu projeto principal.

Para obter instruções detalhadas sobre como configurar funções de execução, consulte:

- Criar um projeto no AWS Device Farm- para configurar funções de execução no nível do projeto
- Criar uma execução de teste no Device Farm- para configurar funções de execução para execuções individuais

Você também pode configurar funções de execução usando a API Device Farm. Para obter mais informações, consulte a Device Farm API Reference.

Práticas recomendadas

Siga estas melhores práticas ao configurar as funções de execução do IAM para seus testes do Device Farm:

- Princípio do menor privilégio: conceda somente as permissões mínimas necessárias para que seus testes funcionem. Evite usar permissões muito amplas, como * ações ou recursos.
- Use permissões específicas de recursos: quando possível, limite as permissões a recursos específicos (por exemplo, buckets específicos do S3 ou tabelas do DynamoDB) em vez de todos os recursos de um tipo.
- Recursos de teste e produção separados: use funções e recursos de teste dedicados para evitar afetar acidentalmente os sistemas de produção durante os testes.
- Revisão regular de funções: revise e atualize periodicamente suas funções de execução para garantir que elas ainda atendam às suas necessidades de teste e sigam as melhores práticas de segurança.
- Use chaves de condição: considere usar chaves de condição do IAM para restringir ainda mais quando e como a função pode ser usada.

Solução de problemas

Se você encontrar problemas com as funções de execução do IAM, verifique o seguinte:

- Relação de confiança: verifique se a política de confiança da função inclui `devicefarm.amazonaws.com` um serviço confiável.
- Permissões: verifique se a função tem as permissões necessárias para os serviços da AWS que seus testes estão tentando acessar.
- Registros de teste: revise os registros de execução do teste em busca de mensagens de erro específicas relacionadas a chamadas de API da AWS ou negações de permissão.

Variáveis de ambiente para ambientes de teste personalizados

O Device Farm configura dinamicamente várias variáveis de ambiente para uso como parte da execução do seu ambiente de teste personalizado.

Tópicos

Variáveis de ambiente personalizadas

O Device Farm suporta a configuração de pares de valores-chave que são aplicados como variáveis de ambiente no host de teste. Elas podem ser configuradas em um projeto Device Farm ou durante a criação da execução; qualquer variável configurada em uma execução substituirá qualquer variável que possa estar configurada em seu projeto principal. As seguintes restrições são aplicáveis:

- Variáveis de ambiente personalizadas não são compatíveis com hosts de teste antigos do iOS. Para obter mais informações, consulte Host de teste iOS antigo.
- Os nomes das variáveis que começam com `$DEVICEFARM_` são reservados para uso interno do serviço.
- Variáveis de ambiente personalizadas não podem ser usadas para configurar a seleção de computação do host de teste em sua especificação de teste.

Variáveis de ambiente comuns

Esta seção descreve as variáveis de ambiente comuns a todos os testes no Device Farm.

$DEVICEFARM_DEVICE_NAME

O dispositivo no qual seus testes são executados. Ele representa o identificador exclusivo do dispositivo (UDID) do dispositivo.

$DEVICEFARM_DEVICE_UDID

O identificador exclusivo do dispositivo.

$DEVICEFARM_DEVICE_PLATFORM_NAME

O nome da plataforma do dispositivo. É `Android` ou `iOS`.

$DEVICEFARM_DEVICE_OS_VERSION

A versão do sistema operacional do dispositivo.

$DEVICEFARM_APP_PATH

(testes de aplicativos móveis)

O caminho do aplicativo para dispositivos móveis na máquina de host onde os testes estão sendo executados. Essa variável não está disponível durante os testes na web.

$DEVICEFARM_LOG_DIR

O caminho para o diretório padrão em que os registros, artefatos e outros arquivos desejados do cliente serão armazenados para recuperação posterior. Usando um exemplo de especificação de teste, 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_PROJECT_ARN

O ARN do projeto principal do trabalho.

$DEVICEFARM_RUN_ARN

O ARN da execução principal do trabalho.

$DEVICEFARM_DEVICE_ARN

O ARN do dispositivo em teste.

$DEVICEFARM_TOTAL_JOBS

O número total de trabalhos associados à execução do Device Farm principal.

$DEVICEFARM_JOB_NUMBER

O número desse trabalho está dentro`$DEVICEFARM_TOTAL_JOBS`. Por exemplo, uma execução pode conter 5 trabalhos e cada um terá um intervalo `$DEVICEFARM_JOB_NUMBER` exclusivo de 0 a 4.

$AWS_REGION

A região da AWS. O serviço definirá isso para corresponder à região na qual o dispositivo em teste está localizado. Ela pode ser substituída por uma variável de ambiente personalizada, se necessário.

$ANDROID_HOME

(Somente para Android)

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

Variáveis de ambiente para testes Appium

Esta seção descreve as variáveis de ambiente usadas por qualquer teste Appium em um ambiente de teste personalizado no Device Farm.

$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR

(Somente para Android)

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

$DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V<N>

(somente iOS)

O caminho de dados derivado de uma versão WebDriverAgent criada para ser executada no Device Farm. A numeração na variável corresponderá à versão principal do WebDriverAgent. Como exemplo, `DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V9` apontará para a WebDriverAgent versão 9.x. Para obter mais informações, consulte Seleção de uma WebDriverAgent versão para testes do iOS.

> **Note**
>
> As variáveis de `$DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V<N>` ambiente só estão presentes em hosts iOS não legados. Para obter mais informações, consulte Host de teste iOS antigo.

`$DEVICEFARM_WDA_DERIVED_DATA_PATH_V9`

(somente iOS, obsoleto)

O caminho de dados derivado de uma versão WebDriverAgent criada para ser executada no Device Farm. Consulte `$DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V<N>` para obter o esquema de nomenclatura de substituição.

Variáveis de ambiente para XCUITest testes

Esta seção descreve as variáveis de ambiente usadas pelo XCUITest teste em um ambiente de teste personalizado no Device Farm.

`$DEVICEFARM_XCUITESTRUN_FILE`

O 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.

`$DEVICEFARM_XCTEST_BUILD_DIRECTORY`

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

Práticas recomendadas para execução de ambientes de teste personalizados

Os tópicos a seguir abordam as melhores práticas recomendadas para usar a execução de testes personalizados com o Device Farm.

Configuração da execução

- Confie no software gerenciado do Device Farm e nos recursos da API para executar a configuração sempre que possível, em vez de aplicar configurações semelhantes por meio de comandos shell no arquivo de especificação de teste. Isso inclui a configuração do host de teste e do dispositivo, pois será mais sustentável e consistente em todos os hosts e dispositivos de teste.

  Embora o Device Farm incentive você a personalizar seu arquivo de especificação de teste o quanto for necessário para executar seus testes, a manutenção do arquivo de especificação de teste pode se tornar difícil com o tempo, à medida que mais comandos personalizados são adicionados a ele. Usando o software gerenciado do Device Farm (por meio de ferramentas como `devicefarm-cli` e as ferramentas padrão disponíveis no`$PATH`) e usando recursos gerenciados (como o parâmetro de `deviceProxy`solicitação) para simplificar o arquivo de especificação de teste, transferindo a responsabilidade da manutenção para o próprio Device Farm.

Especificação de teste e código do pacote de teste

- Não use caminhos absolutos nem confie em versões secundárias específicas em seu arquivo de especificação de teste ou código de pacote de teste. O Device Farm aplica atualizações de rotina ao host de teste selecionado e às versões de software incluídas. Usar caminhos específicos ou absolutos (como `/usr/local/bin/python` em vez de`python`) ou exigir versões secundárias específicas (como Node.js `20.3.1` em vez de apenas `20`) pode fazer com que seus testes não consigam localizar o arquivo /executável necessário.

  Como parte da execução personalizada do teste, o Device Farm configura várias variáveis de ambiente e a `$PATH` variável para garantir que os testes tenham uma experiência consistente em nossos ambientes dinâmicos. Consulte Variáveis de ambiente para ambientes de teste personalizados e Software compatível em ambientes de teste personalizados para obter mais informações.
- Salve os arquivos gerados ou copiados no diretório temporário durante a execução do teste. Hoje, garantimos que o diretório temporário (`/tmp`) esteja acessível ao usuário durante a execução do teste (além dos diretórios gerenciados, como o`$DEVICEFARM_LOG_DIR`). Outros diretórios aos quais o usuário tem acesso podem mudar com o tempo devido às necessidades do serviço ou do sistema operacional em uso.
- Salve seus registros de execução de teste em **`$DEVICEFARM_LOG_DIR`**. Esse é o diretório de artefatos padrão fornecido para sua execução para adicionar logs/artefatos de execução. Cada

um dos exemplos de especificações de teste que fornecemos usa esse diretório para artefatos por padrão.

- Certifique-se de que seus comandos retornem um código diferente de zero em caso de falha durante a `test` fase de sua especificação de teste. Determinamos se sua execução falhou verificando se há um código de saída diferente de zero de cada comando shell invocado durante a `test` fase. Você deve garantir que sua lógica ou estrutura de teste retorne um código de saída diferente de zero para todos os cenários desejados, o que pode exigir configuração adicional.

  Por exemplo, certas estruturas de teste (como JUnit5) não consideram a execução de zero testes uma falha, o que fará com que seus testes sejam detectados como tendo sido executados com êxito, mesmo que nada tenha sido executado. Usando JUnit5 como exemplo, você precisaria especificar a opção de linha de comando `--fail-if-no-tests` para garantir que esse cenário saia com um código de saída diferente de zero.
- Analise a compatibilidade do software com a versão do sistema operacional do dispositivo e a versão do host de teste que você usará para a execução do teste. Por exemplo, há certos recursos nas estruturas de software de teste (por exemplo: Appium) que podem não funcionar conforme o esperado em todas as versões do sistema operacional do dispositivo que está sendo testado.

Segurança

- Evite armazenar ou registrar variáveis confidenciais (como chaves da AWS) em seu arquivo de especificação de teste. Os arquivos da especificação de teste, os scripts gerados pela especificação de teste e os registros do script da especificação de teste são todos fornecidos como artefatos que podem ser baixados no final da execução do teste. Isso pode levar à exposição não intencional de segredos para outros usuários em sua conta com acesso de leitura ao seu teste.

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

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

1. Modo padrão: esse modo de execução de teste foi criado principalmente para fornecer aos clientes relatórios granulares e um ambiente totalmente gerenciado.

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

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

Considerações ao migrar

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

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

   Por outro lado, o modo de execução personalizado não analisa seu pacote de teste; isso significa nenhum pré-processamento e um pós-processamento mínimo para testes ou artefatos de resultados. Isso resulta em tempos totais de end-to-end execução próximos à sua configuração local. Os testes são executados no mesmo formato em que seriam se fossem executados nas máquinas locais. Os resultados dos testes são os mesmos que você obtém localmente e estão disponíveis para download no final da execução do trabalho.
2. Personalização ou flexibilidade: o modo padrão de execução analisa seu pacote de teste para detectar o número de testes e, em seguida, executa cada teste separadamente. Observe que não há garantia de que os testes serão executados na ordem especificada. Como resultado, os testes que exigem uma sequência específica de execução podem não funcionar conforme o esperado. Além disso, não há como personalizar o ambiente da máquina host ou passar arquivos de configuração que possam ser necessários para executar seus testes de uma determinada maneira.

   Por outro lado, o modo personalizado permite que você configure o ambiente da máquina host, incluindo a capacidade de instalar software adicional, passar filtros para seus testes, passar arquivos de configuração e controlar a configuração da execução do teste. Ele consegue isso por meio de um arquivo yaml (também chamado de arquivo testspec) que você pode modificar

adicionando comandos shell a ele. Esse arquivo yaml é convertido em um script de shell que é executado na máquina host de teste. Você pode salvar vários arquivos yaml e escolher um dinamicamente de acordo com seus requisitos ao agendar uma execução.

3. Vídeo ao vivo e registro em log: os modos de execução padrão e personalizado fornecem vídeos e logs para seus testes. No entanto, no modo padrão, você obtém o vídeo e os logs predefinidos de seus testes somente após a conclusão dos testes.

   Por outro lado, o modo personalizado oferece uma transmissão ao vivo do vídeo e dos logs de seus testes do lado do cliente. Além disso, você pode baixar o vídeo e outros artefatos ao final dos testes.

> **Tip**
>
> Se seu caso de uso envolver pelo menos um dos fatores acima, é altamente recomendável mudar para o modo de execução personalizado.

Etapas da migração

Para migrar do modo Padrão para o modo Personalizado, faça o seguinte:

1. Faça login Console de gerenciamento da AWS e abra o console do Device Farm em https://console.aws.amazon.com/devicefarm/.
2. Escolha seu projeto e, em seguida, inicie uma nova execução de automação.
3. Faça upload da aplicação (ou selecione `web app`), escolha o tipo de estrutura de teste, faça upload do pacote de teste e, no parâmetro `Choose your execution environment`, escolha a opção para `Run your test in a custom environment`.
4. Por padrão, o arquivo de exemplo de especificação de teste do Device Farm aparecerá para você visualizar e editar. Esse arquivo de exemplo pode ser usado como ponto de partida para testar seus testes no modo de ambiente personalizado. Depois de verificar se seus testes estão funcionando corretamente no console, você pode alterar qualquer uma de suas integrações de API, CLI e pipeline com o Device Farm para usar esse arquivo de especificação de teste como parâmetro ao programar execuções de teste. Para obter informações sobre como adicionar um arquivo de especificação de teste como parâmetro para suas execuções, consulte a seção do parâmetro `testSpecArn` da API `ScheduleRun` em nosso Guia de API.

Estrutura do Appium

Em um ambiente de teste personalizado, o Device Farm não insere nem substitui nenhum recurso do Appium em seus testes da estrutura do Appium. Você deve especificar os recursos do Appium do teste no arquivo YAML da especificação de teste ou no código de teste.

Instrumentação do Android

Você não precisa fazer alterações a fim de mover os testes de instrumentação do Android para um ambiente de teste personalizado.

iOS XCUITest

Você não precisa fazer alterações para mover seus XCUITest testes do iOS para um ambiente de teste personalizado.

Extensão de ambientes de teste personalizados no Device Farm

O AWS Device Farm permite configurar um ambiente personalizado para testes automatizados (modo personalizado), que é a abordagem recomendada para todos os usuários do Device Farm. O modo personalizado do Device Farm permite que você execute mais do que apenas seu pacote de testes. Nesta seção, você aprenderá como estender sua suíte de testes e otimizar seus testes.

Para acessar mais informações sobre ambientes de teste personalizados no Device Farm, consulte Ambiente de teste personalizado no AWS Device Farm..

Tópicos

- Configurar o PIN de um dispositivo ao executar testes no Device Farm
- Acelerar os testes baseados no Appium nos recursos desejados do Device Farm
- Usando Webhooks e outros APIs após a execução dos testes no Device Farm
- Adicionar arquivos extras ao seu pacote de teste no Device Farm

Configurar o PIN de um dispositivo ao executar testes no Device Farm

Algumas aplicações exigem que você defina um PIN no dispositivo. O Device Farm não é compatível com a configuração nativa de um PIN em dispositivos. No entanto, isso é possível com as seguintes ressalvas:

- O dispositivo deve estar executando o Android 8 ou superior.
- O PIN deve ser removido após a conclusão do teste.

Para definir o PIN em seus testes, use as fases `pre_test` e `post_test` para definir e remover o PIN, conforme mostrado a seguir:

```
phases:
    pre_test:
      - # ... among your pre_test commands
      - DEVICE_PIN_CODE="1234"
      - adb shell locksettings set-pin "$DEVICE_PIN_CODE"
    post_test:
      - # ... Among your post_test commands
      - adb shell locksettings clear --old "$DEVICE_PIN_CODE"
```

Quando o conjunto de testes é iniciado, o PIN 1234 é definido. Depois que sua suíte de testes sair, o PIN será removido.

> **Warning**
>
> Se você não remover o PIN do dispositivo após a conclusão do teste, o dispositivo e sua conta serão colocados em quarentena.

Para conhecer mais maneiras de estender seu pacote de testes e otimizar seus testes, consulte Extensão de ambientes de teste personalizados no Device Farm.

Acelerar os testes baseados no Appium nos recursos desejados do Device Farm

Ao usar o Appium, você pode descobrir que o conjunto de testes do modo padrão é muito lento. Isso ocorre porque o Device Farm aplica as configurações padrão e não faz nenhuma suposição sobre como você deseja usar o ambiente Appium. Embora esses padrões sejam criados com base nas melhores práticas do setor, eles podem não se aplicar à sua situação. Para ajustar os parâmetros do servidor Appium, você pode ajustar os recursos padrão do Appium em sua especificação de teste. Por exemplo, o seguinte define o recurso `usePrebuildWDA` como `true` em um conjunto de testes do iOS para acelerar o tempo de início inicial:

```
phases:
  pre_test:
    - # ... Start up Appium
    - >-
    appium --log-timestamp
    --default-capabilities "{\"usePrebuiltWDA\": true, \"derivedDataPath\":
\"$DEVICEFARM_WDA_DERIVED_DATA_PATH\",
    \"deviceName\": \"$DEVICEFARM_DEVICE_NAME\", \"platformName\":
\"$DEVICEFARM_DEVICE_PLATFORM_NAME\", \"app\":\"$DEVICEFARM_APP_PATH\",
    \"automationName\":\"XCUITest\", \"udid\":\"$DEVICEFARM_DEVICE_UDID_FOR_APPIUM\",
 \"platformVersion\":\"$DEVICEFARM_DEVICE_OS_VERSION\"}"
    >> $DEVICEFARM_LOG_DIR/appiumlog.txt 2>&1 &
```

Os recursos do Appium devem ser uma estrutura JSON citada e com escape de shell.

Os seguintes recursos do Appium são fontes comuns de melhorias de desempenho:

`noReset` e `fullReset`

Esses dois recursos, que são mutuamente exclusivos, descrevem o comportamento do Appium após a conclusão de cada sessão. Quando `noReset` está definido como `true`, o servidor Appium não remove dados da aplicação quando uma sessão do Appium termina, efetivamente não fazendo nenhuma limpeza. `fullReset` desinstala e limpa todos os dados da aplicação do dispositivo após o encerramento da sessão. Para obter mais informações, consulte Reset Strategies na documentação do Appium.

`ignoreUnimportantViews` (somente Android)

Instrui o Appium a compactar a hierarquia da interface do usuário do Android somente em visualizações relevantes para o teste, acelerando as pesquisas de determinados elementos. No entanto, isso pode interromper alguns conjuntos de testes XPath baseados porque a hierarquia do layout da interface do usuário foi alterada.

`skipUnlock` (somente Android)

Informa à Appium que não há um código PIN definido atualmente, o que acelera os testes após um evento de desligamento da tela ou outro evento de bloqueio.

`webDriverAgentUrl` (somente iOS)

Instrui o Appium a assumir que uma dependência essencial do iOS, `webDriverAgent`, já está em execução e disponível para aceitar solicitações HTTP no URL especificado. Se `webDriverAgent` ainda não estiver instalado e funcionando, o Appium pode levar algum tempo no início de um conjunto de testes para iniciar o `webDriverAgent`. Se você iniciar `webDriverAgent` por conta própria e definir `webDriverAgentUrl` como `http://localhost:8100` ao iniciar o Appium, poderá inicializar o conjunto de testes mais rapidamente. Observe que esse recurso nunca deve ser usado junto com o recurso `useNewWDA`.

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

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

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

Depois, você pode adicionar o código a seguir ao arquivo de especificação de teste para garantir que `webDriverAgent` tenha iniciado com êxito. Esse código deve ser executado no final da fase de pré-teste depois de garantir que o Appium tenha sido iniciado com sucesso:

```
      # Wait for WebDriverAgent to start
      - >-
        start_wda_timeout=0;
        while [ true ];
        do
          if [ $start_wda_timeout -gt 60 ];
```

```
          then
              echo "WebDriverAgent server never started in 60 seconds.";
              exit 1;
          fi;
          grep -i "ServerURLHere" $DEVICEFARM_LOG_DIR/webdriveragent_log.txt >> /
dev/null 2>&1;
          if [ $? -eq 0 ];
          then
              echo "WebDriverAgent REST http interface listener started";
              break;
          else
              echo "Waiting for WebDriverAgent server to start. Sleeping for 1
 seconds";
              sleep 1;
              start_wda_timeout=$((start_wda_timeout+1));
          fi;
        done;
```

Para obter mais informações sobre os recursos compatíveis com o Appium, consulte Appium Desired Capabilities na documentação do Appium.

Para conhecer mais maneiras de estender seu pacote de testes e otimizar seus testes, consulte Extensão de ambientes de teste personalizados no Device Farm.

Usando Webhooks e outros APIs após a execução dos testes no Device Farm

Você pode fazer com que o Device Farm chame um webhook depois que cada conjunto de testes terminar de usar curl. O processo para fazer isso varia de acordo com o destino e a formatação. Para seu webhook específico, consulte a documentação desse webhook. O exemplo a seguir publica uma mensagem sempre que uma suíte de testes termina em um webhook do Slack:

```
phases:
  post_test:
    - curl -X POST -H 'Content-type: application/json' --data '{"text":"Tests on
 '$DEVICEFARM_DEVICE_NAME' have finished!"}' https://hooks.slack.com/services/
T00000000/B00000000/XXXXXXXXXXXXXXXXXXXXXXXX
```

Para obter mais informações sobre como usar webhooks com o Slack, consulte Sending your first Slack message using Webhook na referência de API do Slack.

Para conhecer mais maneiras de estender seu pacote de testes e otimizar seus testes, consulte Extensão de ambientes de teste personalizados no Device Farm.

Você não está limitado a usar curl para chamar webhooks. Os pacotes de teste podem incluir scripts e ferramentas extras, desde que sejam compatíveis com o ambiente de execução do Device Farm. Por exemplo, seu pacote de teste pode incluir scripts auxiliares que fazem solicitações a outros APIs. Certifique-se de que todos os pacotes necessários estejam instalados junto com os requisitos da sua suíte de testes. Para adicionar um script que seja executado após a conclusão da suíte de testes, inclua o script em seu pacote de teste e adicione o seguinte à sua especificação de teste:

```
phases:
  post_test:
    - python post_test.py
```

> **Note**
>
> A manutenção de todas as chaves de API ou outros tokens de autenticação usados em seu pacote de teste é de sua responsabilidade. Recomendamos que você mantenha qualquer forma de credencial de segurança fora do controle de origem, use credenciais com o menor número possível de privilégios e use tokens revogáveis e de curta duração sempre que possível. Para verificar os requisitos de segurança, consulte a documentação do terceiro APIs que você usa.

Se você planeja usar AWS serviços como parte de sua suíte de execução de testes, você deve usar credenciais temporárias do IAM, geradas fora da suíte de testes e incluídas no pacote de teste. Essas credenciais devem ter o menor número de permissões concedidas e a menor vida útil possível. Para obter mais informações sobre a criação de credenciais temporárias, consulte Requesting temporary security credentials no Guia do usuário do IAM.

Para conhecer mais maneiras de estender seu pacote de testes e otimizar seus testes, consulte Extensão de ambientes de teste personalizados no Device Farm.

Adicionar arquivos extras ao seu pacote de teste no Device Farm

Talvez você queira usar arquivos adicionais como parte de seus testes como arquivos extras de configuração ou dados de teste adicionais. Você pode incluir esses arquivos adicionais ao seu pacote de testes antes de carregá-lo no AWS Device Farm e acessá-los no modo de ambiente

personalizado. Basicamente, todos os formatos de upload de pacotes de teste (ZIP, IPA, APK, JAR etc.) são formatos de arquivamento de pacotes compatíveis com operações ZIP padrão.

Você pode adicionar arquivos ao seu arquivo de teste antes de enviá-lo AWS Device Farm usando o seguinte comando:

```
$ zip zip-with-dependencies.zip extra_file
```

Para um diretório de arquivos extras:

```
$ zip -r zip-with-dependencies.zip extra_files/
```

Esses comandos funcionam conforme o esperado para todos os formatos de upload de pacotes de teste, exceto para arquivos IPA. Para arquivos IPA, especialmente quando usados com XCUITests, recomendamos que você coloque os arquivos extras em um local um pouco diferente devido à forma como os pacotes de teste do AWS Device Farm iOS são resignados. Ao criar seu teste do iOS, o diretório do aplicativo de teste estará localizado dentro de outro diretório chamado*Payload*.

Por exemplo, é assim que um desses diretórios de teste do iOS pode parecer:

```
$ tree
.
### Payload
    ### ADFiOSReferenceAppUITests-Runner.app
        ### ADFiOSReferenceAppUITests-Runner
        ### Frameworks
        #   ### XCTAutomationSupport.framework
        #   #   ### Info.plist
        #   #   ### XCTAutomationSupport
        #   #   ### _CodeSignature
        #   #   #   ### CodeResources
        #   #   ### version.plist
        #   ### XCTest.framework
        #       ### Info.plist
        #       ### XCTest
        #       ### _CodeSignature
        #       #   ### CodeResources
        #       ### en.lproj
        #       #   ### InfoPlist.strings
        #       ### version.plist
        ### Info.plist
```

```
        ### PkgInfo
        ### PlugIns
        #   ### ADFiOSReferenceAppUITests.xctest
        #   #   ### ADFiOSReferenceAppUITests
        #   #   ### Info.plist
        #   #   ### _CodeSignature
        #   #       ### CodeResources
        #   ### ADFiOSReferenceAppUITests.xctest.dSYM
        #       ### Contents
        #           ### Info.plist
        #           ### Resources
        #               ### DWARF
        #                   ### ADFiOSReferenceAppUITests
        ### _CodeSignature
        #   ### CodeResources
        ### embedded.mobileprovision
```

Para esses XCUITest pacotes, adicione qualquer arquivo extra ao diretório que termina *.app* dentro do *Payload* diretório. Por exemplo, os comandos a seguir mostram como você pode adicionar um arquivo a esse pacote de teste:

```
$ mv extra_file Payload/*.app/
$ zip -r my_xcui_tests.ipa Payload/
```

Ao adicionar um arquivo ao seu pacote de teste, você pode esperar um comportamento de interação um pouco diferente no AWS Device Farm com base no formato de upload. Se o upload usou a extensão de arquivo ZIP, AWS Device Farm descompactará automaticamente o upload antes do teste e deixará os arquivos descompactados no local com a *$DEVICEFARM_TEST_PACKAGE_PATH* variável de ambiente. (Isso significa que se você adicionasse um arquivo chamado *extra_file* à raiz do arquivo, como no primeiro exemplo, ele estaria localizado *$DEVICEFARM_TEST_PACKAGE_PATH/extra_file* durante o teste).

Para usar um exemplo mais prático, se você for um usuário do Appium TestNG que deseja incluir um *testng.xml* arquivo com seu teste, você pode incluí-lo em seu arquivo usando o seguinte comando:

```
$ zip zip-with-dependencies.zip testng.xml
```

Em seguida, você pode alterar seu comando de teste no modo de ambiente personalizado para o seguinte:

```
java -D appium.screenshots.dir=$DEVICEFARM_SCREENSHOT_PATH org.testng.TestNG -testjar
 *-tests.jar -d $DEVICEFARM_LOG_DIR/test-output $DEVICEFARM_TEST_PACKAGE_PATH/
testng.xml
```

Se a extensão de upload do pacote de teste não for ZIP (por exemplo, arquivo APK, IPA ou JAR), o arquivo do pacote enviado em si será encontrado em*$DEVICEFARM_TEST_PACKAGE_PATH*. Como esses ainda são arquivos em formato de arquivamento, você pode descompactar o arquivo para acessar os arquivos adicionais de dentro. Por exemplo, o comando a seguir descompactará o conteúdo do pacote de teste (para arquivos APK, IPA ou JAR) no */tmp* diretório:

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

No caso de um arquivo APK ou JAR, você encontraria seus arquivos extras descompactados no */tmp* diretório (por exemplo,*/tmp/extra_file*). No caso de um arquivo IPA, conforme explicado anteriormente, os arquivos extras estariam em um local ligeiramente diferente dentro da pasta que termina em*.app*, que está dentro do *Payload* diretório. Por exemplo, com base no exemplo do IPA acima, o arquivo seria encontrado no local */tmp/Payload/ADFiOSReferenceAppUITests-Runner.app/extra_file* (referenciável como). */tmp/Payload/*.app/extra_file*

Para conhecer mais maneiras de estender seu pacote de testes e otimizar seus testes, consulte Extensão de ambientes de teste personalizados no Device Farm.

Acesso remoto no AWS Device Farm

O acesso remoto permite que você deslize o dedo, faça gestos e interaja com um dispositivo por meio de um navegador da web em tempo real para testar a funcionalidade e reproduzir os problemas dos clientes. Você interage com um dispositivo específico criando uma sessão de acesso remoto com esse dispositivo.

Uma sessão no Device Farm é uma interação em tempo real com um dispositivo físico real hospedado em um navegador da web. Uma sessão exibe o dispositivo específico que você selecionou ao iniciar a sessão. Um usuário pode iniciar mais de uma sessão por vez, mas o número total de dispositivos simultâneos está restrito ao número de slots para dispositivo que você tem. Você pode comprar slots de dispositivos com base na família de dispositivos (dispositivos Android ou iOS). Para obter mais informações, consulte Definição de preço do Device Farm.

Atualmente, o Device Farm oferece um subconjunto de dispositivos para testes de acesso remoto. Novos dispositivos são adicionados ao grupo de dispositivos sempre.

O Device Farm captura vídeo de cada sessão de acesso remoto e gera logs de atividade durante a sessão. Esses resultados incluem todas as informações que você fornece durante uma sessão.

> **Note**
>
> Por motivos de segurança, recomendamos evitar fornecer ou inserir informações confidenciais, como números de conta, informações pessoais de login e outros detalhes durante uma sessão de acesso remoto. Se possível, use alternativas desenvolvidas especificamente para testes, como contas de teste.

Tópicos

Criar 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 remota

Console

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e, em seguida, Projetos.
3. Se você já tiver um projeto, selecione-o na lista. Caso contrário, crie um projeto seguindo as instruções em Criar um projeto no AWS Device Farm.
4. Na guia Acesso remoto, escolha Criar sessão de acesso remoto.
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.
6. (Opcional) Inclua um aplicativo e aplicativos auxiliares como parte da sessão. Eles podem ser aplicativos recém-carregados ou aplicativos enviados anteriormente neste projeto nos últimos 30 dias (após 30 dias, os uploads de aplicativos expirarão).
7. Em Session name (Nome da sessão), insira um nome para a sessão.
8. Escolha Confirm and start session (Confirmar e iniciar sessão).

AWS CLI

Observação: essas instruções se concentram somente na criação de uma sessão de acesso remoto. Para obter instruções sobre como fazer upload de um aplicativo para uso durante sua sessão, consulte Como automatizar o upload de aplicativos.

Primeiro, verifique se sua versão da AWS CLI é baixando e up-to-date instalando a versão mais recente.

> **Important**
>
> Alguns comandos mencionados neste documento não estão disponíveis em versões mais antigas da AWS CLI.

Em seguida, você pode determinar em qual dispositivo você gostaria de testar:

```
$ aws devicefarm list-devices
```

Isso mostrará uma saída como a seguinte:

```
{
    "devices":
    [
        {
            "arn": "arn:aws:devicefarm:us-
west-2::device:DE5BD47FF3BD42C3A14BF7A6EFB1BFE7",
            "name": "Google Pixel 8",
            "remoteAccessEnabled": true,
            "availability": "HIGHLY_AVAILABLE"
            ...
        },
        ...
    ]
}
```

Em seguida, você pode criar sua sessão de acesso remoto com um ARN de dispositivo de sua escolha:

```
$ aws devicefarm create-remote-access-session \
  --project-arn arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef \
  --device-arn arn:aws:devicefarm:us-west-2::device:DE5BD47FF3BD42C3A14BF7A6EFB1BFE7
 \
  --app-arn arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
 \
```

```
  --configuration '{
      "auxiliaryApps": [
          "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
          "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
      ]
  }'
```

Isso mostrará uma saída como a seguinte:

```
{
    "remoteAccessSession": {
        "arn": "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000",
        "name": "Google Pixel 8",
        "status": "PENDING",
        ...
}
```

Agora, opcionalmente, podemos fazer uma pesquisa e esperar que a sessão esteja pronta:

```
$ POLL_INTERVAL=3
TIMEOUT=600
DEADLINE=$(( $(date +%s) + TIMEOUT ))

while [[ "$(date +%s)" -lt "$DEADLINE" ]]; do

  STATUS=$(aws devicefarm get-remote-access-session \
    --arn "$DEVICE_FARM_SESSION_ARN" \
    --query 'remoteAccessSession.status' \
    --output text)

  case "$STATUS" in
    RUNNING)
      echo "Session is ready with status: $STATUS"
      break
      ;;
    STOPPING|COMPLETED)
      echo "Session ended early with status: $STATUS"
      exit 1
      ;;
```

```
  esac

done
```

Python

Observação: essas instruções se concentram somente na criação de uma sessão de acesso remoto. Para obter instruções sobre como fazer upload de um aplicativo para uso durante sua sessão, consulte Como automatizar o upload de aplicativos.

Este exemplo primeiro encontra qualquer dispositivo Google Pixel disponível no Device Farm, depois cria uma sessão de acesso remoto com ele e espera até que a sessão seja executada.

```
import random
import time
import boto3

client = boto3.client("devicefarm", region_name="us-west-2")

# 1) Gather all matching devices via paginated ListDevices with filters
filters = [
    {"attribute": "MODEL",        "operator": "CONTAINS", "values": ["Pixel"]},
    {"attribute": "AVAILABILITY", "operator": "EQUALS",   "values": ["AVAILABLE"]},
]

matching_arns = []
next_token = None
while True:
    args = {"filters": filters}
    if next_token:
        args["nextToken"] = next_token
    page = client.list_devices(**args)
    for d in page.get("devices", []):
        matching_arns.append(d["arn"])
    next_token = page.get("nextToken")
    if not next_token:
        break

if not matching_arns:
    raise RuntimeError("No available Google Pixel device found.")

# Randomly select one device from the full matching set
device_arn = random.choice(matching_arns)
```

```
print("Selected device ARN:", device_arn)

# 2) Create remote access session and wait until RUNNING
resp = client.create_remote_access_session(
    projectArn="arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef",
    deviceArn=device_arn,
    appArn="arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
  # optional
    configuration={
        "auxiliaryApps": [  # optional
            "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
            "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
        ]
    },
)

session_arn = resp["remoteAccessSession"]["arn"]
print(f"Created Remote Access Session: {session_arn}")

poll_interval = 3
timeout = 600
deadline = time.time() + timeout
terminal_states = ["STOPPING", "COMPLETED"]

while True:
    out = client.get_remote_access_session(arn=session_arn)
    status = out["remoteAccessSession"]["status"]
    print(f"Current status: {status}")

    if status == "RUNNING":
        print(f"Session is ready with status: {status}")
        break
    if status in terminal_states:
        raise RuntimeError(f"Session ended early with status: {status}")
    if time.time() >= deadline:
        raise RuntimeError("Timed out waiting for session to be ready.")
    time.sleep(poll_interval)
```

Java

Observação: essas instruções se concentram somente na criação de uma sessão de acesso remoto. Para obter instruções sobre como fazer upload de um aplicativo para uso durante sua sessão, consulte Como automatizar o upload de aplicativos.

Observação: este exemplo usa o AWS SDK for Java v2 e é compatível com as versões 11 e superiores do JDK.

Este exemplo primeiro encontra qualquer dispositivo Google Pixel disponível no Device Farm, depois cria uma sessão de acesso remoto com ele e espera até que a sessão seja executada.

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.concurrent.ThreadLocalRandom;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import
 software.amazon.awssdk.services.devicefarm.model.CreateRemoteAccessSessionConfiguration;
import
 software.amazon.awssdk.services.devicefarm.model.CreateRemoteAccessSessionRequest;
import
 software.amazon.awssdk.services.devicefarm.model.CreateRemoteAccessSessionResponse;
import software.amazon.awssdk.services.devicefarm.model.Device;
import software.amazon.awssdk.services.devicefarm.model.DeviceFilter;
import software.amazon.awssdk.services.devicefarm.model.DeviceFilterAttribute;
import
 software.amazon.awssdk.services.devicefarm.model.GetRemoteAccessSessionRequest;
import
 software.amazon.awssdk.services.devicefarm.model.GetRemoteAccessSessionResponse;
import software.amazon.awssdk.services.devicefarm.model.ListDevicesRequest;
import software.amazon.awssdk.services.devicefarm.model.ListDevicesResponse;
import software.amazon.awssdk.services.devicefarm.model.RuleOperator;

public class CreateRemoteAccessSession {
  public static void main(String[] args) throws Exception {
    DeviceFarmClient client = DeviceFarmClient.builder()
        .region(Region.US_WEST_2)
        .build();

    String projectArn = "arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef";
```

```
    String appArn     = "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
    String aux1       = "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
    String aux2       = "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789

    // 1) Gather all matching devices via paginated ListDevices with filters
    List<DeviceFilter> filters = Arrays.asList(
        DeviceFilter.builder()
            .attribute(DeviceFilterAttribute.MODEL)
            .operator(RuleOperator.CONTAINS)
            .values("Pixel")
            .build(),
        DeviceFilter.builder()
            .attribute(DeviceFilterAttribute.AVAILABILITY)
            .operator(RuleOperator.EQUALS)
            .values("AVAILABLE")
            .build()
    );

    List<String> matchingDeviceArns = new ArrayList<>();
    String next = null;
    do {
      ListDevicesResponse page = client.listDevices(
          ListDevicesRequest.builder().filters(filters).nextToken(next).build());
      for (Device d : page.devices()) {
        matchingDeviceArns.add(d.arn());
      }
      next = page.nextToken();
    } while (next != null);

    if (matchingDeviceArns.isEmpty()) {
      throw new RuntimeException("No available Google Pixel device found.");
    }

    // Randomly select one device from the full matching set
    String deviceArn = matchingDeviceArns.get(
        ThreadLocalRandom.current().nextInt(matchingDeviceArns.size()));
    System.out.println("Selected device ARN: " + deviceArn);

    // 2) Create Remote Access session and wait until it is RUNNING
    CreateRemoteAccessSessionConfiguration cfg =
 CreateRemoteAccessSessionConfiguration.builder()
```

```
        .auxiliaryApps(Arrays.asList(aux1, aux2))
        .build();

    CreateRemoteAccessSessionResponse res = client.createRemoteAccessSession(
        CreateRemoteAccessSessionRequest.builder()
            .projectArn(projectArn)
            .deviceArn(deviceArn)
            .appArn(appArn)       // optional
            .configuration(cfg)   // optional
            .build());

    String sessionArn = res.remoteAccessSession().arn();
    System.out.println("Created Remote Access Session: " + sessionArn);

    int pollIntervalMs = 3000;
    long timeoutMs = 600_000L;
    long deadline = System.currentTimeMillis() + timeoutMs;

    while (true) {
      GetRemoteAccessSessionResponse get = client.getRemoteAccessSession(
          GetRemoteAccessSessionRequest.builder().arn(sessionArn).build());
      String status = get.remoteAccessSession().statusAsString();
      System.out.println("Current status: " + status);

      if ("RUNNING".equals(status)) {
        System.out.println("Session is ready with status: " + status);
        break;
      }
      if ("STOPPING".equals(status) || "COMPLETED".equals(status)) {
        throw new RuntimeException("Session ended early with status: " + status);
      }
      if (System.currentTimeMillis() >= deadline) {
        throw new RuntimeException("Timed out waiting for session to be ready.");
      }
      Thread.sleep(pollIntervalMs);
    }
  }
}
```

JavaScript

Observação: essas instruções se concentram somente na criação de uma sessão de acesso remoto. Para obter instruções sobre como fazer upload de um aplicativo para uso durante sua sessão, consulte Como automatizar o upload de aplicativos.

Observação: este exemplo usa o AWS SDK para JavaScript v3.

Este exemplo primeiro encontra qualquer dispositivo Google Pixel disponível no Device Farm, depois cria uma sessão de acesso remoto com ele e espera até que a sessão seja executada.

```
import {
  DeviceFarmClient,
  ListDevicesCommand,
  CreateRemoteAccessSessionCommand,
  GetRemoteAccessSessionCommand,
} from "@aws-sdk/client-device-farm";

const client = new DeviceFarmClient({ region: "us-west-2" });

// 1) Gather all matching devices via paginated ListDevices with filters
const filters = [
  { attribute: "MODEL",        operator: "CONTAINS", values: ["Pixel"] },
  { attribute: "AVAILABILITY", operator: "EQUALS",   values: ["AVAILABLE"] },
];

let nextToken;
const matching = [];

while (true) {
  const page = await client.send(new ListDevicesCommand({ filters, nextToken }));
  for (const d of page.devices ?? []) {
    matching.push(d.arn);
  }
  nextToken = page.nextToken;
  if (!nextToken) break;
}

if (matching.length === 0) {
  throw new Error("No available Google Pixel device found.");
}

// Randomly select one device from the full matching set
```

```
const deviceArn = matching[Math.floor(Math.random() * matching.length)];
console.log("Selected device ARN:", deviceArn);

// 2) Create remote access session and wait until RUNNING
const out = await client.send(new CreateRemoteAccessSessionCommand({
  projectArn: "arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef",
  deviceArn,
  appArn: "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
 optional
  configuration: {
    auxiliaryApps: [ // optional
      "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
      "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
    ],
  },
}));

const sessionArn = out.remoteAccessSession?.arn;
console.log("Created Remote Access Session:", sessionArn);

const pollIntervalMs = 3000;
const timeoutMs = 600000;
const deadline = Date.now() + timeoutMs;

while (true) {
  const get = await client.send(new GetRemoteAccessSessionCommand({ arn:
 sessionArn }));
  const status = get.remoteAccessSession?.status;
  console.log("Current status:", status);

  if (status === "RUNNING") {
    console.log("Session is ready with status:", status);
    break;
  }
  if (status === "STOPPING" || status === "COMPLETED") {
    throw new Error(`Session ended early with status: ${status}`);
  }
  if (Date.now() >= deadline) {
    throw new Error("Timed out waiting for session to be ready.");
  }
```

```
  await new Promise((r) => setTimeout(r, pollIntervalMs));
}
```

C#

Observação: essas instruções se concentram somente na criação de uma sessão de acesso remoto. Para obter instruções sobre como fazer upload de um aplicativo para uso durante sua sessão, consulte Como automatizar o upload de aplicativos.

Este exemplo primeiro encontra qualquer dispositivo Google Pixel disponível no Device Farm, depois cria uma sessão de acesso remoto com ele e espera até que a sessão seja executada.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

class Program
{
    static async Task Main()
    {
        var client = new AmazonDeviceFarmClient(RegionEndpoint.USWest2);

        // 1) Gather all matching devices via paginated ListDevices with filters
        var filters = new List<DeviceFilter>
        {
            new DeviceFilter { Attribute = DeviceAttribute.MODEL,        Operator =
 RuleOperator.CONTAINS, Values = new List<string>{ "Pixel" } },
            new DeviceFilter { Attribute = DeviceAttribute.AVAILABILITY, Operator =
 RuleOperator.EQUALS,   Values = new List<string>{ "AVAILABLE" } },
        };

        var matchingArns = new List<string>();
        string nextToken = null;

        do
        {
            var list = await client.ListDevicesAsync(new ListDevicesRequest
            {
                Filters = filters,
                NextToken = nextToken
```

```
            });

            foreach (var d in list.Devices)
                matchingArns.Add(d.Arn);

            nextToken = list.NextToken;
        }
        while (nextToken != null);

        if (matchingArns.Count == 0)
            throw new Exception("No available Google Pixel device found.");

        // Randomly select one device from the full matching set
        var rnd = new Random();
        var deviceArn = matchingArns[rnd.Next(matchingArns.Count)];
        Console.WriteLine($"Selected device ARN: {deviceArn}");

        // 2) Create remote access session and wait until RUNNING
        var request = new CreateRemoteAccessSessionRequest
        {
            ProjectArn = "arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef",
            DeviceArn  = deviceArn,
            AppArn     = "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
 optional
            Configuration = new CreateRemoteAccessSessionConfiguration
            {
                AuxiliaryApps = new List<string>
                {
                    "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
                    "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
                }
            }
        };

        request.Configuration.AuxiliaryApps.RemoveAll(string.IsNullOrWhiteSpace);

        var response = await client.CreateRemoteAccessSessionAsync(request);
        var sessionArn = response.RemoteAccessSession.Arn;
        Console.WriteLine($"Created Remote Access Session: {sessionArn}");
```

```
        var pollIntervalMs = 3000;
        var timeoutMs = 600000;
        var deadline = DateTime.UtcNow.AddMilliseconds(timeoutMs);

        while (true)
        {
            var get = await client.GetRemoteAccessSessionAsync(new
 GetRemoteAccessSessionRequest { Arn = sessionArn });
            var status = get.RemoteAccessSession.Status.Value;
            Console.WriteLine($"Current status: {status}");

            if (status == "RUNNING")
            {
                Console.WriteLine($"Session is ready with status: {status}");
                break;
            }
            if (status == "STOPPING" || status == "COMPLETED")
            {
                throw new Exception($"Session ended early with status: {status}");
            }
            if (DateTime.UtcNow >= deadline)
            {
                throw new TimeoutException("Timed out waiting for session to be
 ready.");
            }

            await Task.Delay(pollIntervalMs);
        }
    }
}
```

Ruby

Observação: essas instruções se concentram somente na criação de uma sessão de acesso remoto. Para obter instruções sobre como fazer upload de um aplicativo para uso durante sua sessão, consulte Como automatizar o upload de aplicativos.

Este exemplo primeiro encontra qualquer dispositivo Google Pixel disponível no Device Farm, depois cria uma sessão de acesso remoto com ele e espera até que a sessão seja executada.

```
require 'aws-sdk-devicefarm'

client = Aws::DeviceFarm::Client.new(region: 'us-west-2')
```

```
# 1) Gather all matching devices via paginated ListDevices with filters
filters = [
  { attribute: 'MODEL',        operator: 'CONTAINS', values: ['Pixel'] },
  { attribute: 'AVAILABILITY', operator: 'EQUALS',   values: ['AVAILABLE'] },
]

matching_arns = []
next_token = nil
loop do
  resp = client.list_devices(filters: filters, next_token: next_token)
  resp.devices&.each { |d| matching_arns << d.arn }
  next_token = resp.next_token
  break unless next_token
end

abort "No available Google Pixel device found." if matching_arns.empty?

# Randomly select one device from the full matching set
device_arn = matching_arns.sample
puts "Selected device ARN: #{device_arn}"

# 2) Create remote access session and wait until RUNNING
resp = client.create_remote_access_session(
  project_arn: "arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef",
  device_arn:  device_arn,
  app_arn:     "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
 # optional
  configuration: {
    auxiliary_apps: [ # optional
      "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
      "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
    ].compact
  }
)

session_arn = resp.remote_access_session.arn
puts "Created Remote Access Session: #{session_arn}"

poll_interval = 3
```

```
timeout = 600
deadline = Time.now + timeout
terminal = %w[STOPPING COMPLETED]

loop do
  get = client.get_remote_access_session(arn: session_arn)
  status = get.remote_access_session.status
  puts "Current status: #{status}"

  if status == 'RUNNING'
    puts "Session is ready with status: #{status}"
    break
  end

  abort "Session ended early with status: #{status}" if terminal.include?(status)
  abort "Timed out waiting for session to be ready." if Time.now >= deadline
  sleep poll_interval
end
```

Próximas etapas

O Device Farm inicia a sessão assim que o dispositivo e a infraestrutura solicitados estão disponíveis, normalmente em alguns minutos. A caixa de diálogo Dispositivo solicitado ficará aberta até o momento em que a sessão iniciar. Para cancelar a solicitação de sessão, escolha Cancel request (Cancelar solicitação).

Se o dispositivo selecionado estiver indisponível ou ocupado, o status da sessão será exibido como Dispositivo pendente, indicando que talvez seja necessário aguardar algum tempo até que o dispositivo esteja disponível para teste.

Se sua conta atingiu o limite de simultaneidade para dispositivos públicos com ou sem medição, o status da sessão será exibido como Simultaneidade pendente. Para slots de dispositivos ilimitados, você pode aumentar a simultaneidade comprando mais slots de dispositivos. Para pay-as-you-go dispositivos medidos, entre em contato com a AWS por meio de um ticket de suporte para solicitar um aumento na cota de serviço.

Quando a configuração da sessão começa, ela mostra primeiro um status de Em andamento e, em seguida, um status de Conexão enquanto seu navegador local tenta abrir uma conexão remota com o dispositivo.

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.

Após o início da sessão, você pode interagir com o dispositivo no navegador da web ou testar o dispositivo usando o Appium.

Usar uma sessão de acesso remoto no AWS Device Farm

Para obter informações sobre como executar testes interativos de aplicativos Android e iOS por meio de sessões de acesso remoto, consulte Sessões.

- Pré-requisitos
- Use uma sessão no console do Device Farm
- Próximas etapas
- Dicas e truques

Pré-requisitos

- Crie uma sessão. Siga as instruções em Criar uma sessão e retorne para esta página.

Use uma sessão no console do Device Farm

Assim que o dispositivo que você solicitou para uma sessão de acesso remoto ficar disponível, o console exibirá a tela do dispositivo. A sessão tem duração máxima de 150 minutos. O tempo restante na sessão aparece no campo Tempo restante próximo ao nome do dispositivo.

Instalar um aplicativo

Para instalar uma aplicação no dispositivo de sessão, em Instalar aplicações, selecione Escolher arquivo e escolha o arquivo .apk (Android) ou o arquivo .ipa (iOS) que você deseja instalar. Os aplicativos que você executa em uma sessão de acesso remoto não exigem nenhum teste de instrumentação nem provisionamento.

> **Note**
>
> Ao fazer upload de um aplicativo, às vezes o aplicativo demora um pouco para ficar disponível. Uma mensagem de confirmação aparecerá informando se o aplicativo foi instalado com sucesso ou não.

Controlar o dispositivo

Você pode interagir com o dispositivo exibido no console assim como faria com um dispositivo físico real, usando o mouse ou um dispositivo semelhante para tocar e o teclado na tela do dispositivo. Para dispositivos Android, existem botões em View controls (Visualizar controles) que funcionam como os botões Home (Início) e Back (Voltar) em um dispositivo Android. Para dispositivos iOS, existe um botão Home (Início) que funciona da mesma forma que o botão de início em um dispositivo iOS. Você também pode alternar aplicações executadas no dispositivo escolhendo Aplicações recentes.

Alternar modos retrato e paisagem

Você também pode alternar os modos retrato (vertical) e paisagem (horizontal) nos dispositivos que está usando.

Próximas etapas

O Device Farm continua a sessão até que você a interrompa manualmente ou até que o limite de tempo de 150 minutos seja atingido. Para encerrar a sessão, escolha o botão Interromper sessão. Depois que a sessão for interrompida, você poderá acessar o vídeo que foi capturado e os logs que foram gerados. Para obter mais informações, consulte Recuperando os resultados da sessão.

Dicas e truques

Você pode ter problemas de desempenho com a sessão de acesso remoto em algumas AWS regiões. Em parte, isso se deve à latência em algumas regiões. Se tiver problemas de desempenho, permita que a sessão de acesso remota recupere o atraso para então interagir novamente com o aplicativo.

Recuperar os 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 Usar uma sessão de acesso remoto no AWS Device Farm e retorne para esta página.

Visualização de detalhes da sessão

Quando uma sessão de acesso remoto termina, o console do Device Farm exibe uma tabela que contém detalhes sobre a atividade durante a sessão. Para obter mais informações, consulte Analisar informações do log.

Para retornar aos detalhes de uma sessão em um momento posterior:

1. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e, em seguida, Projetos.
2. Escolha o projeto que contém a sessão.
3. Selecione Acesso remoto e escolha a sessão que você deseja revisar na lista.

Download de vídeo ou logs de sessão

Quando uma sessão de acesso remoto termina, o console do Device Farm fornece acesso a uma captura de vídeo da sessão e aos logs de atividade. Nos resultados da sessão, escolha a guia Files (Arquivos) para obter uma lista de links para o vídeo e os logs da sessão. Você pode visualizar esses arquivos no navegador ou os salvar localmente.

Teste do Appium no AWS Device Farm

Durante uma sessão de acesso remoto, você pode executar testes Appium em seu ambiente local, visando o dispositivo da sessão usando um endpoint Appium gerenciado. Com um endpoint Appium, você pode desenvolver, testar e executar o código Appium com feedback rápido e iteração rápida. Essa abordagem de teste do lado do cliente oferece a flexibilidade de se conectar a um dispositivo Device Farm a partir de qualquer ambiente cliente Appium de sua escolha.

Para complementar os testes do lado do cliente, o Device Farm também oferece suporte à execução de testes na infraestrutura gerenciada pelo serviço, chamada de execução do lado do servidor. Nessa abordagem, você pode fazer upload do aplicativo e dos testes para o serviço e, em seguida, executá-los paralelamente em vários dispositivos usando hosts de teste gerenciados pelo serviço. Essa abordagem se adapta bem para testes em vários dispositivos de forma independente, bem como para testes a partir do contexto de um CI/CD pipeline.

Para saber mais sobre a execução no lado do servidor, consulte. Frameworks de teste e testes integrados

Tópicos

O que é um endpoint Appium?

O Appium é uma popular estrutura de teste de software de código aberto para testar aplicativos web nativos, híbridos e móveis em diferentes dispositivos, incluindo telefones celulares e tablets, para iOS e Android. Ele permite que desenvolvedores e engenheiros de QA (Garantia de Qualidade) escrevam scripts que possam controlar remotamente um dispositivo, simular interações com o usuário e verificar se o aplicativo em teste está se comportando conforme o esperado. O Appium interage com os aplicativos da perspectiva de um usuário final, permitindo que os testadores desenvolvam testes que simulem como usuários reais usarão o aplicativo para seus testes.

O Appium é construído no modelo cliente-servidor, em que um cliente local solicita que um servidor Appium (local ou remoto) comande um dispositivo em seu nome. O servidor Appium gerencia um driver para comunicação com o dispositivo, como o UIAutomator2 driver para Android ou o driver XCUITest para iOS. Todos os comandos seguem os WebDriver padrões do W3C sobre como controlar um dispositivo.

O endpoint Appium do Device Farm expõe uma URL do servidor Appium para o dispositivo em sua sessão de acesso remoto. O URL do endpoint Appium será específico para esse dispositivo nessa sessão e permanecerá válido durante a sessão, permitindo que você itere no mesmo dispositivo sem tempo adicional de configuração. Para obter mais informações sobre o acesso remoto, consulteAcesso remoto.

Começando com os testes do Appium

Para a maioria dos usuários do Appium, o uso do Device Farm para testes do Appium requer apenas pequenas alterações na configuração de teste existente.

Em um alto nível, há três etapas para usar o Device Farm para testes Appium do lado do cliente:

1. Primeiro, você precisa criar uma sessão de acesso remoto para testar um dispositivo Device Farm. Você pode incluir seus aplicativos como parte de sua solicitação de acesso remoto ou instalar aplicativos após o início da sessão.
2. Quando a sessão estiver em execução, você pode copiar o URL do endpoint Appium e usá-lo por meio de uma ferramenta independente (como o Appium Inspector) ou do código de teste do Appium em seu IDE. O URL será válido durante a sessão de acesso remoto.
3. E, finalmente, depois que o teste do Appium for iniciado, você poderá revisar os registros do servidor Appium ao vivo durante a execução do teste junto com o stream de vídeo do seu dispositivo.

Interagindo com o dispositivo usando o Appium

Depois de criar uma sessão de acesso remoto, o dispositivo estará disponível para testes do Appium. Durante toda a sessão de acesso remoto, você pode executar quantas sessões do Appium quiser no dispositivo, sem limites de quais clientes você usa. Por exemplo, você pode começar executando um teste usando o código Appium local do seu IDE e, em seguida, passar a usar o Appium Inspector para solucionar quaisquer problemas encontrados. A sessão pode durar até 150

minutos, no entanto, se não houver atividade por mais de 5 minutos (por meio do console interativo ou do endpoint Appium), a sessão expirará.

Usando aplicativos para testar com sua sessão do Appium

O Device Farm permite que você use seus aplicativos como parte da solicitação de criação da sessão de acesso remoto ou instale aplicativos durante a própria sessão de acesso remoto. Esses aplicativos são instalados automaticamente no dispositivo em teste e são injetados como recursos padrão para qualquer solicitação de sessão do Appium. Ao criar uma sessão de acesso remoto, você tem a opção de passar um ARN do aplicativo, que será usado por padrão como o `appium:app` recurso para todas as sessões subsequentes do Appium, bem como o aplicativo auxiliar ARNs, que será usado como o recurso. `appium:otherApps`

Por exemplo, se você criar uma sessão de acesso remoto usando um aplicativo `com.aws.devicefarm.sample` como seu aplicativo e `com.aws.devicefarm.other.sample` como um de seus aplicativos auxiliares, ao criar uma sessão do Appium, ela terá recursos semelhantes aos seguintes:

```
{
    "value":
    {
        "sessionId": "abcdef123456-1234-5678-abcd-abcdef123456",
        "capabilities":
        {
            "app": "/tmp/com.aws.devicefarm.sample.apk",
            "otherApps": "[\"/tmp/com.aws.devicefarm.other.sample.apk\"]",
            ...
        }
    }
}
```

Durante sua sessão, você pode instalar aplicativos adicionais (no console ou usando a `InstallToRemoteAccessSession`API). Eles substituirão todos os aplicativos existentes usados anteriormente como `appium:app` recurso. Se esses aplicativos usados anteriormente tiverem um nome de pacote distinto, eles permanecerão no dispositivo e serão usados como parte do `appium:otherApps` recurso.

Por exemplo, se você usar inicialmente um aplicativo `com.aws.devicefarm.sample` ao criar sua sessão de acesso remoto, mas depois instalar um novo aplicativo chamado

`com.aws.devicefarm.other.sample` durante a sessão, suas sessões do Appium terão recursos semelhantes aos seguintes:

```
{
    "value":
    {
        "sessionId": "abcdef123456-1234-5678-abcd-abcdef123456",
        "capabilities":
        {
            "app": "/tmp/com.aws.devicefarm.other.sample.apk",
            "otherApps": "[\"/tmp/com.aws.devicefarm.sample.apk\"]",
            ...
        }
    }
}
```

Se preferir, você pode especificar explicitamente os recursos do seu aplicativo usando o nome do aplicativo (usando os `appium:bundleId` recursos `appium:appPackage` ou para Android e iOS, respectivamente).

Se você estiver testando um aplicativo web, especifique a `browserName` capacidade da sua solicitação de criação de sessão do Appium. O `Chrome` navegador está disponível em todos os dispositivos Android e o `Safari` navegador está disponível em todos os dispositivos iOS.

O Device Farm não suporta a transmissão de uma URL remota ou um caminho do sistema de arquivos local `appium:app` durante uma sessão de acesso remoto. Faça upload de aplicativos para o Device Farm e, em vez disso, inclua-os na sessão.

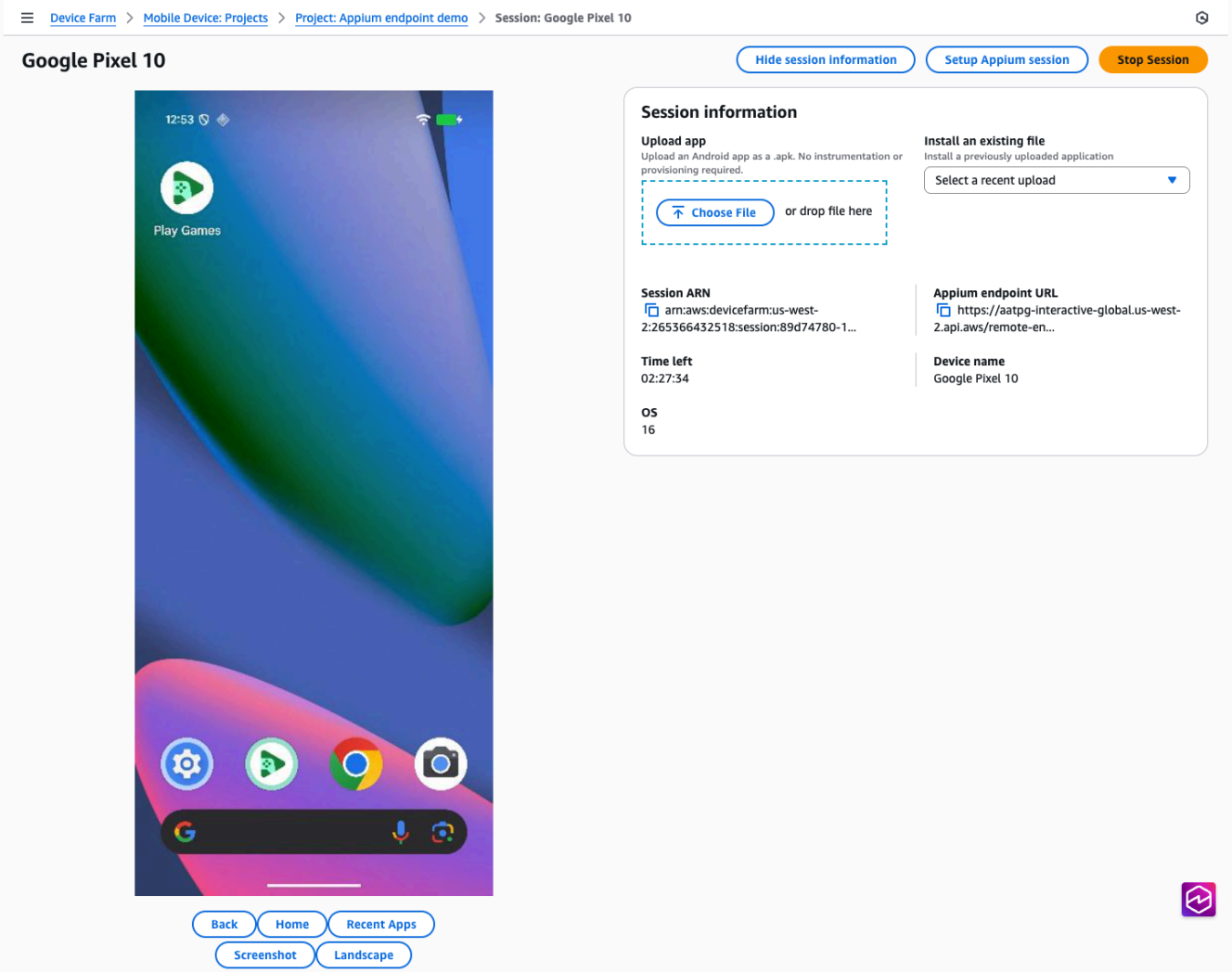
> **Note**
>
> Para obter mais informações sobre o upload automático de aplicativos como parte de sua sessão de acesso remoto, consulte Como automatizar o upload de aplicativos.

Como usar o endpoint Appium

Aqui estão as etapas para acessar o endpoint Appium da sessão a partir do console AWS CLI, do e. AWS SDKs Essas etapas incluem como começar a executar testes usando várias estruturas de teste de clientes da Appium:

Console

1. Abra sua página de sessão de acesso remoto em seu navegador da web:



2. Para executar uma sessão usando o Appium Inspector, faça o seguinte:

   a. Clique no botão Configurar sessão do Appium

   b. Siga as instruções na página sobre como iniciar uma sessão usando o Appium Inspector.

3. Para executar um teste Appium a partir do seu IDE local, faça o seguinte:

   a. Clique no ícone “copiar” ao lado do texto URL do endpoint Appium

   b. Cole esse URL em seu código Appium local onde quer que você especifique atualmente seu endereço remoto ou executor de comando. Para exemplos específicos do idioma, clique em uma das guias nesta janela de exemplo para o idioma de sua escolha.

AWS CLI

Primeiro, verifique se sua versão da AWS CLI é baixando e up-to-date instalando a versão mais recente.

> **Important**
>
> O campo de endpoint do Appium não está disponível em versões mais antigas da AWS CLI.

Quando sua sessão estiver em execução, o URL do endpoint do Appium estará disponível por meio de um campo nomeado `remoteDriverEndpoint` na resposta a uma chamada para a API: `GetRemoteAccessSession`

```
$ aws devicefarm get-remote-access-session \
    --arn "arn:aws:devicefarm:us-west-2:123456789876:session:abcdef123456-1234-5678-abcd-abcdef123456/abcdef123456-1234-5678-abcd-abcdef123456/00000"
```

Isso mostrará uma saída como a seguinte:

```
{
    "remoteAccessSession": {
        "arn": "arn:aws:devicefarm:us-west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/abcdef123456-1234-5678-abcd-abcdef123456/00000",
        "name": "Google Pixel 8",
        "status": "RUNNING",
        "endpoints": {
            "remoteDriverEndpoint": "https://devicefarm-interactive-global.us-west-2.api.aws/remote-endpoint/ABCD1234...",
        ...
}
```

Você pode usar essa URL em seu código Appium local onde quer que você especifique atualmente seu endereço remoto ou executor de comando. Para exemplos específicos do idioma, clique em uma das guias nesta janela de exemplo para o idioma de sua escolha.

Para ver um exemplo de como interagir com o endpoint diretamente da linha de comando, você pode usar a ferramenta de linha de comando curl para chamar um endpoint diretamente: WebDriver

```
$ curl "https://devicefarm-interactive-global.us-west-2.api.aws/remote-endpoint/
ABCD1234.../status"
```

Isso mostrará uma saída como a seguinte:

```
{
    "value":
    {
        "ready": true,
        "message": "The server is ready to accept new connections",
        "build":
        {
            "version": "2.5.1"
        }
    }
}
```

Python

Quando sua sessão estiver em execução, o URL do endpoint do Appium estará disponível por meio de um campo nomeado `remoteDriverEndpoint` na resposta a uma chamada para a API: GetRemoteAccessSession

```
# To get the URL
import sys
import boto3
from botocore.exceptions import ClientError

def get_appium_endpoint() -> str:
    session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000"
    device_farm_client = boto3.client("devicefarm", region_name="us-west-2")

    try:
        resp = device_farm_client.get_remote_access_session(arn=session_arn)
    except ClientError as exc:
        sys.exit(f"Failed to call Device Farm: {exc}")

    remote_access_session = resp.get("remoteAccessSession", {})
    endpoints = remote_access_session.get("endpoints", {})
    endpoint = endpoints.get("remoteDriverEndpoint")
```

```
    if not endpoint:
        sys.exit("Device Farm response did not include
 endpoints.remoteDriverEndpoint")

    return endpoint

# To use the URL
from appium import webdriver
from appium.options.android import UiAutomator2Options

opts = UiAutomator2Options()
driver = webdriver.Remote(get_appium_endpoint(), options=opts)
# ...
driver.quit()
```

Java

Observação: este exemplo usa o AWS SDK for Java v2 e é compatível com as versões 11 e superiores do JDK.

Quando sua sessão estiver em execução, o URL do endpoint do Appium estará disponível por meio de um campo nomeado `remoteDriverEndpoint` na resposta a uma chamada para a API: `GetRemoteAccessSession`

```
// To get the URL
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import
 software.amazon.awssdk.services.devicefarm.model.GetRemoteAccessSessionRequest;
import
 software.amazon.awssdk.services.devicefarm.model.GetRemoteAccessSessionResponse;

public class AppiumEndpointBuilder {
    public static String getAppiumEndpoint() throws Exception {
        String session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000";

        try (DeviceFarmClient client = DeviceFarmClient.builder()
                .region(Region.US_WEST_2)
                .credentialsProvider(DefaultCredentialsProvider.create())
```

```
                .build()) {

            GetRemoteAccessSessionResponse resp = client.getRemoteAccessSession(
                    GetRemoteAccessSessionRequest.builder().arn(session_arn).build()
            );

            String endpoint =
 resp.remoteAccessSession().endpoints().remoteDriverEndpoint();
            if (endpoint == null || endpoint.isEmpty()) {
                throw new IllegalStateException("remoteDriverEndpoint missing from
 response");
            }
            return endpoint;
        }
    }
}

// To use the URL
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.options.UiAutomator2Options;

import java.net.URL;

public class ExampleTest {
    public static void main(String[] args) throws Exception {
        String endpoint = AppiumEndpointBuilder.getAppiumEndpoint();
        UiAutomator2Options options = new UiAutomator2Options();
        AndroidDriver driver = new AndroidDriver(new URL(endpoint), options);

        try {
            // ... your test ...
        } finally {
            driver.quit();
        }
    }
}
```

JavaScript

Observação: este exemplo usa AWS SDK para JavaScript v3 e WebDriverIO v8+ usando o Node 18+.

Quando sua sessão estiver em execução, o URL do endpoint do Appium estará disponível por meio de um campo nomeado `remoteDriverEndpoint` na resposta a uma chamada para a API: GetRemoteAccessSession

```
// To get the URL
import { DeviceFarmClient, GetRemoteAccessSessionCommand } from "@aws-sdk/client-
device-farm";

export async function getAppiumEndpoint() {
  const sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000";

  const client = new DeviceFarmClient({ region: "us-west-2" });
  const resp = await client.send(new GetRemoteAccessSessionCommand({ arn:
 sessionArn }));

  const endpoint = resp?.remoteAccessSession?.endpoints?.remoteDriverEndpoint;
  if (!endpoint) throw new Error("remoteDriverEndpoint missing from response");
  return endpoint;
}

// To use the URL with WebdriverIO
import { remote } from "webdriverio";

(async () => {
  const endpoint = await getAppiumEndpoint();
  const u = new URL(endpoint);

  const driver = await remote({
    protocol: u.protocol.replace(":", ""),
    hostname: u.hostname,
    port: u.port ? Number(u.port) : (u.protocol === "https:" ? 443 : 80),
    path: u.pathname + u.search,
    capabilities: {
      platformName: "Android",
      "appium:automationName": "UiAutomator2",
      // ...other caps...
    },
  });

  try {
    // ... your test ...
```

```
  } finally {
    await driver.deleteSession();
  }
})();
```

C#

Quando sua sessão estiver em execução, o URL do endpoint do Appium estará disponível por meio de um campo nomeado `remoteDriverEndpoint` na resposta a uma chamada para a API: [GetRemoteAccessSession]

```
// To get the URL
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

public static class AppiumEndpointBuilder
{
    public static async Task<string> GetAppiumEndpointAsync()
    {
        var sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000";

        var config = new AmazonDeviceFarmConfig
        {
            RegionEndpoint = RegionEndpoint.USWest2
        };
        using var client = new AmazonDeviceFarmClient(config);

        var resp = await client.GetRemoteAccessSessionAsync(new
 GetRemoteAccessSessionRequest { Arn = sessionArn });
        var endpoint = resp?.RemoteAccessSession?.Endpoints?.RemoteDriverEndpoint;

        if (string.IsNullOrWhiteSpace(endpoint))
            throw new InvalidOperationException("RemoteDriverEndpoint missing from
 response");

        return endpoint;
    }
}
```

```
// To use the URL
using OpenQA.Selenium.Appium;
using OpenQA.Selenium.Appium.Android;

class Example
{
    static async Task Main()
    {
        var endpoint = await AppiumEndpointBuilder.GetAppiumEndpointAsync();

        var options = new AppiumOptions();
        options.PlatformName = "Android";
        options.AutomationName = "UiAutomator2";

        using var driver = new AndroidDriver(new Uri(endpoint), options);
        try
        {
            // ... your test ...
        }
        finally
        {
            driver.Quit();
        }
    }
}
```

Ruby

Quando sua sessão estiver em execução, o URL do endpoint do Appium estará disponível por meio de um campo nomeado `remoteDriverEndpoint` na resposta a uma chamada para a API: `GetRemoteAccessSession`

```
# To get the URL
require 'aws-sdk-devicefarm'

def get_appium_endpoint
  session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000"

  client = Aws::DeviceFarm::Client.new(region: 'us-west-2')
  resp = client.get_remote_access_session(arn: session_arn)
```

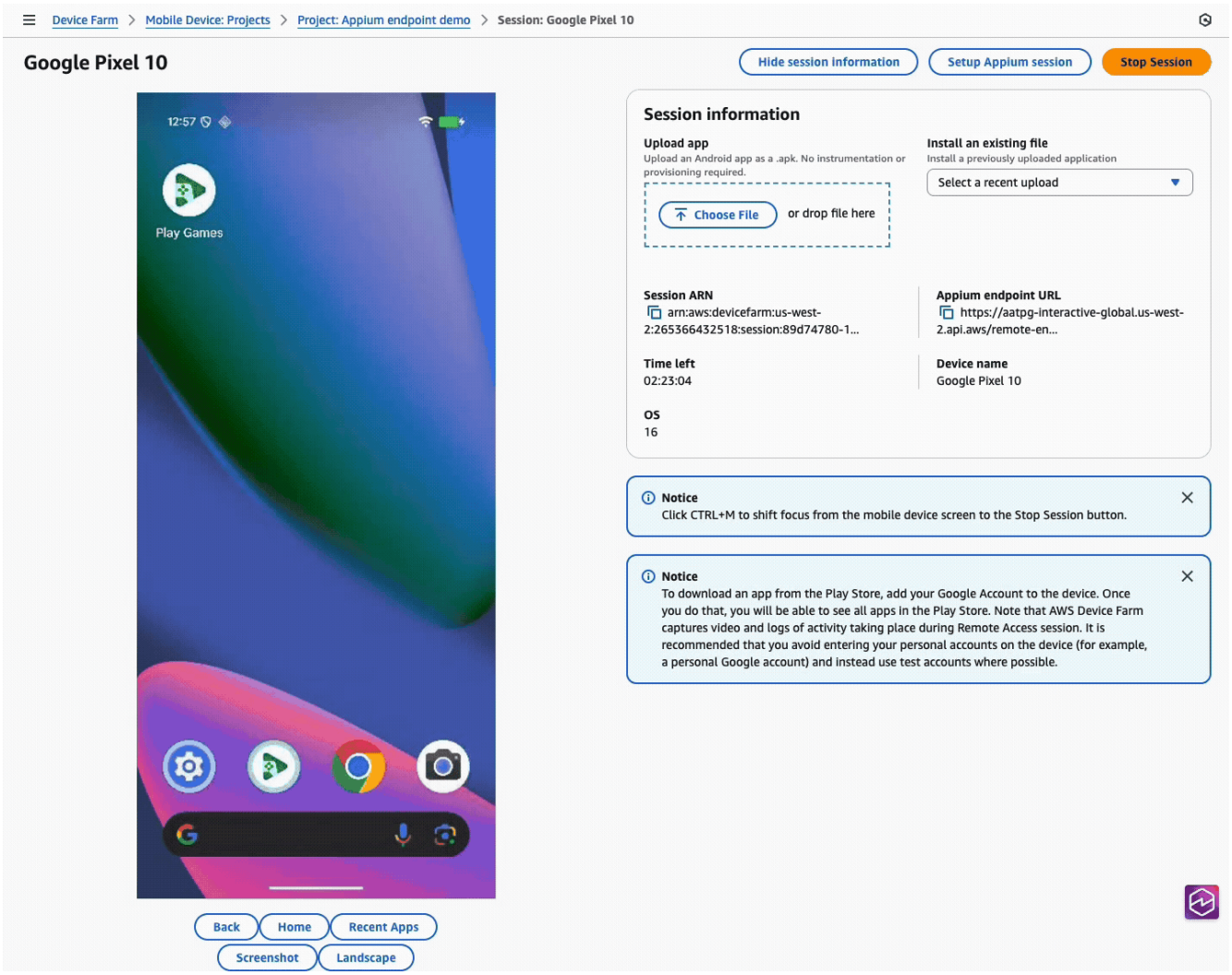
```
  endpoint = resp.remote_access_session.endpoints.remote_driver_endpoint
  raise "remote_driver_endpoint missing from response" if endpoint.nil? ||
 endpoint.empty?
  endpoint
end

# To use the URL
require 'appium_lib_core'

endpoint = get_appium_endpoint
opts = {
  server_url: endpoint,
  capabilities: {
    'platformName' => 'Android',
    'appium:automationName' => 'UiAutomator2'
  }
}

driver = Appium::Core.for(opts).start_driver
begin
  # ... your test ...
ensure
  driver.quit
end
```

Revisando os registros do servidor Appium

Depois de iniciar uma sessão do Appium, você pode ver os registros do servidor Appium ao vivo no console do Device Farm ou baixá-los após o término da sessão de acesso remoto. Aqui estão as instruções para fazer isso:

Console

1. No console do Device Farm, abra a sessão de acesso remoto do seu dispositivo.
2. Inicie uma sessão de endpoint Appium com o dispositivo a partir do seu IDE local ou do Appium Inspector
3. Em seguida, o log do servidor Appium aparecerá ao lado do dispositivo na página da sessão de acesso remoto, com as “informações da sessão” disponíveis na parte inferior da página abaixo do dispositivo:



AWS CLI

Observação: este exemplo usa a ferramenta de linha de comando *curl* para extrair o log do Device Farm.

Durante ou após a sessão, você pode usar a `ListArtifacts`API do Device Farm para baixar o log do servidor Appium.

```
$ aws devicefarm list-artifacts \
  --type FILE \
  --arn arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678
```

Isso mostrará uma saída como a seguinte durante a sessão:

```
{
    "artifacts": [
        {
            "arn": "arn:aws:devicefarm:us-
west-2:111122223333:artifact:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-4567
            "name": "AppiumServerLogOutput",
            "type": "APPIUM_SERVER_LOG_OUTPUT",
            "extension": "",
            "url": "https://prod-us-west-2-results.s3.dualstack.us-
west-2.amazonaws.com/111122223333/12345678..."
        }
    ]
}
```

E o seguinte após o término da sessão:

```
{
    "artifacts": [
        {
            "arn": "arn:aws:devicefarm:us-
west-2:111122223333:artifact:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-4567
            "name": "Appium Server Output",
            "type": "APPIUM_SERVER_OUTPUT",
            "extension": "log",
            "url": "https://prod-us-west-2-results.s3.dualstack.us-
west-2.amazonaws.com/111122223333/12345678..."
        }
    ]
}
```

```
$ curl "https://prod-us-west-2-results.s3.dualstack.us-
west-2.amazonaws.com/111122223333/12345678..."
```

Isso mostrará uma saída como a seguinte:

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1',
info Appium   allowInsecure:
info Appium    [ 'execute_driver_script',
info Appium      'session_discovery',
info Appium      'perf_record',
```

```
info Appium      'adb_shell',
info Appium      'chromedriver_autodownload',
info Appium      'get_server_logs' ],
info Appium   keepAliveTimeout: 0,
info Appium   logNoColors: true,
info Appium   logTimestamp: true,
info Appium   longStacktrace: true,
info Appium   sessionOverride: true,
info Appium   strictCaps: true,
info Appium   useDrivers: [ 'uiautomator' ] }
```

Python

Observação: este exemplo usa o *`requests`* pacote de terceiros para baixar o log, bem como o AWS SDK para Python*`boto3`*.

Durante ou após a sessão, você pode usar a `ListArtifacts`API do Device Farm para recuperar a URL de log do servidor Appium e, em seguida, baixá-la.

```
import pathlib
import requests
import boto3

def download_appium_log():
    session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678
    client = boto3.client("devicefarm", region_name="us-west-2")

    # 1) List artifacts for the session (FILE artifacts), handling pagination
    artifacts = []
    token = None
    while True:
        kwargs = {"arn": session_arn, "type": "FILE"}
        if token:
            kwargs["nextToken"] = token
        resp = client.list_artifacts(**kwargs)
        artifacts.extend(resp.get("artifacts", []))
        token = resp.get("nextToken")
        if not token:
            break

    if not artifacts:
        raise RuntimeError("No artifacts found in this session")
```

```
    # Filter strictly to Appium server logs
    allowed = {"APPIUM_SERVER_OUTPUT", "APPIUM_SERVER_LOG_OUTPUT"}
    filtered = [a for a in artifacts if a.get("type") in allowed]
    if not filtered:
        raise RuntimeError("No Appium server log artifacts found (expected
 APPIUM_SERVER_OUTPUT or APPIUM_SERVER_LOG_OUTPUT)")

    # Prefer the final 'OUTPUT' log, else the live 'LOG_OUTPUT'
    chosen = (next((a for a in filtered if a.get("type") == "APPIUM_SERVER_OUTPUT"),
 None)
              or next((a for a in filtered if a.get("type") ==
 "APPIUM_SERVER_LOG_OUTPUT"), None))

    url = chosen["url"]
    ext = chosen.get("extension") or "log"
    out = pathlib.Path(f"./appium_server_log.{ext}")

    # 2) Download the artifact
    with requests.get(url, stream=True) as r:
        r.raise_for_status()
        with open(out, "wb") as fh:
            for chunk in r.iter_content(chunk_size=1024 * 1024):
                if chunk:
                    fh.write(chunk)

    print(f"Saved Appium server log to: {out.resolve()}")

download_appium_log()
```

Isso mostrará uma saída como a seguinte:

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', allowInsecure: [ 'execute_driver_script', ... ],
 useDrivers: [ 'uiautomator' ] }
```

Java

Observação: este exemplo usa o AWS SDK for Java v2 *`HttpClient`* e para baixar o log, e é compatível com as versões 11 e superiores do JDK.

Durante ou após a sessão, você pode usar a [ListArtifacts](#)API do Device Farm para recuperar a URL de log do servidor Appium e, em seguida, baixá-la.

```
import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.nio.file.Path;
import java.time.Duration;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import software.amazon.awssdk.services.devicefarm.model.Artifact;
import software.amazon.awssdk.services.devicefarm.model.ArtifactCategory;
import software.amazon.awssdk.services.devicefarm.model.ListArtifactsRequest;
import software.amazon.awssdk.services.devicefarm.model.ListArtifactsResponse;

public class AppiumLogDownloader {

    public static void main(String[] args) throws Exception {
        String sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678

        try (DeviceFarmClient client = DeviceFarmClient.builder()
                .region(Region.US_WEST_2)
                .build()) {

            // 1) List artifacts for the session (FILE artifacts) with pagination
            List<Artifact> all = new ArrayList<>();
            String token = null;
            do {
                ListArtifactsRequest.Builder b = ListArtifactsRequest.builder()
                        .arn(sessionArn)
                        .type(ArtifactCategory.FILE);
                if (token != null) b.nextToken(token);
                ListArtifactsResponse page = client.listArtifacts(b.build());
                all.addAll(page.artifacts());
                token = page.nextToken();
            } while (token != null && !token.isBlank());
```

```
            // Filter strictly to Appium logs
            List<Artifact> filtered = all.stream()
                    .filter(a -> {
                        String t = a.typeAsString();
                        return "APPIUM_SERVER_OUTPUT".equals(t) ||
"APPIUM_SERVER_LOG_OUTPUT".equals(t);
                    })
                    .toList();

            if (filtered.isEmpty()) {
                throw new RuntimeException("No Appium server log artifacts found
(expected APPIUM_SERVER_OUTPUT or APPIUM_SERVER_LOG_OUTPUT).");
            }

            // Prefer OUTPUT; else LOG_OUTPUT
            Artifact chosen = filtered.stream()
                    .filter(a -> "APPIUM_SERVER_OUTPUT".equals(a.typeAsString()))
                    .findFirst()
                    .orElseGet(() -> filtered.stream()
                            .filter(a ->
"APPIUM_SERVER_LOG_OUTPUT".equals(a.typeAsString()))
                            .findFirst()
                            .get());

            String url = chosen.url();
            String ext = (chosen.extension() == null ||
chosen.extension().isBlank()) ? "log" : chosen.extension();
            Path out = Path.of("appium_server_log." + ext);

            // 2) Download the artifact with HttpClient
            HttpClient http = HttpClient.newBuilder()
                    .connectTimeout(Duration.ofSeconds(10))
                    .build();

            HttpRequest get = HttpRequest.newBuilder(URI.create(url))
                    .timeout(Duration.ofMinutes(5))
                    .GET()
                    .build();

            HttpResponse<Path> resp = http.send(get,
HttpResponse.BodyHandlers.ofFile(out));
            if (resp.statusCode() / 100 != 2) {
```

```
                throw new IOException("Failed to download log, HTTP " +
 resp.statusCode());
            }
            System.out.println("Saved Appium server log to: " +
 out.toAbsolutePath());
        }
    }
}
```

Isso mostrará uma saída como a seguinte:

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', ..., useDrivers: [ 'uiautomator' ] }
```

JavaScript

Observação: este exemplo usa o AWS SDK para JavaScript (v3) e o Node 18+ *`fetch`* para baixar o log.

Durante ou após a sessão, você pode usar a `ListArtifacts`API do Device Farm para recuperar a URL de log do servidor Appium e, em seguida, baixá-la.

```
import { DeviceFarmClient, ListArtifactsCommand } from "@aws-sdk/client-device-
farm";
import { createWriteStream } from "fs";
import { pipeline } from "stream";
import { promisify } from "util";

const pipe = promisify(pipeline);
const client = new DeviceFarmClient({ region: "us-west-2" });

const sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678

// 1) List artifacts for the session (FILE artifacts), handling pagination
const artifacts = [];
let nextToken;
do {
  const page = await client.send(new ListArtifactsCommand({
    arn: sessionArn,
    type: "FILE",
```

```
    nextToken
  }));
  artifacts.push(...(page.artifacts ?? []));
  nextToken = page.nextToken;
} while (nextToken);

if (!artifacts.length) throw new Error("No artifacts found");

// Strict filter to Appium logs
const filtered = (artifacts ?? []).filter(a =>
  a.type === "APPIUM_SERVER_OUTPUT" || a.type === "APPIUM_SERVER_LOG_OUTPUT"
);
if (!filtered.length) {
  throw new Error("No Appium server log artifacts found (expected
 APPIUM_SERVER_OUTPUT or APPIUM_SERVER_LOG_OUTPUT).");
}

// Prefer OUTPUT; else LOG_OUTPUT
const chosen =
  filtered.find(a => a.type === "APPIUM_SERVER_OUTPUT") ??
  filtered.find(a => a.type === "APPIUM_SERVER_LOG_OUTPUT");

const url = chosen.url;
const ext = chosen.extension || "log";
const outPath = `./appium_server_log.${ext}`;

// 2) Download the artifact
const resp = await fetch(url);
if (!resp.ok) {
  throw new Error(`Failed to download log: ${resp.status} ${await
 resp.text().catch(()=>"")}`);
}
await pipe(resp.body, createWriteStream(outPath));
console.log("Saved Appium server log to:", outPath);
```

Isso mostrará uma saída como a seguinte:

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', allowInsecure: [ 'execute_driver_script', ... ],
 useDrivers: [ 'uiautomator' ] }
```

C#

Observação: este exemplo usa o AWS SDK for .NET *`HttpClient`* e para baixar o log.

Durante ou após a sessão, você pode usar a `ListArtifacts`API do Device Farm para recuperar a URL de log do servidor Appium e, em seguida, baixá-la.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Net.Http;
using System.Threading.Tasks;
using System.Linq;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

class AppiumLogDownloader
{
    static async Task Main()
    {
        var sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678

        using var client = new AmazonDeviceFarmClient(RegionEndpoint.USWest2);

        // 1) List artifacts for the session (FILE artifacts), handling pagination
        var all = new List<Artifact>();
        string? token = null;
        do
        {
            var page = await client.ListArtifactsAsync(new ListArtifactsRequest
            {
                Arn = sessionArn,
                Type = ArtifactCategory.FILE,
                NextToken = token
            });
            if (page.Artifacts != null) all.AddRange(page.Artifacts);
            token = page.NextToken;
        } while (!string.IsNullOrEmpty(token));

        if (all.Count == 0)
            throw new Exception("No artifacts found");
```

```
        // Strict filter to Appium logs
        var filtered = all.Where(a =>
            a.Type == "APPIUM_SERVER_OUTPUT" || a.Type ==
 "APPIUM_SERVER_LOG_OUTPUT").ToList();

        if (filtered.Count == 0)
            throw new Exception("No Appium server log artifacts found (expected
 APPIUM_SERVER_OUTPUT or APPIUM_SERVER_LOG_OUTPUT).");

        // Prefer OUTPUT; else LOG_OUTPUT
        var chosen = filtered.FirstOrDefault(a => a.Type == "APPIUM_SERVER_OUTPUT")
                    ?? filtered.First(a => a.Type == "APPIUM_SERVER_LOG_OUTPUT");

        var url = chosen.Url;
        var ext = string.IsNullOrWhiteSpace(chosen.Extension) ? "log" :
 chosen.Extension;
        var outPath = $"./appium_server_log.{ext}";

        // 2) Download the artifact
        using var http = new HttpClient();
        using var resp = await http.GetAsync(url,
 HttpCompletionOption.ResponseHeadersRead);
        resp.EnsureSuccessStatusCode();
        await using (var fs = File.Create(outPath))
        {
            await resp.Content.CopyToAsync(fs);
        }
        Console.WriteLine($"Saved Appium server log to:
 {Path.GetFullPath(outPath)}");
    }
}
```

Isso mostrará uma saída como a seguinte:

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', ..., useDrivers: [ 'uiautomator' ] }
```

Ruby

Observação: este exemplo usa o AWS SDK for *`Net::HTTP`* Ruby e para baixar o log.

Durante ou após a sessão, você pode usar a [ListArtifacts]API do Device Farm para recuperar a URL de log do servidor Appium e, em seguida, baixá-la.

```
require "aws-sdk-devicefarm"
require "net/http"
require "uri"

client = Aws::DeviceFarm::Client.new(region: "us-west-2")
session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678

# 1) List artifacts for the session (FILE artifacts), handling pagination
artifacts = []
token = nil
loop do
  page = client.list_artifacts(arn: session_arn, type: "FILE", next_token: token)
  artifacts.concat(page.artifacts || [])
  token = page.next_token
  break if token.nil? || token.empty?
end

raise "No artifacts found" if artifacts.empty?

# Strict filter to Appium logs
filtered = (artifacts || []).select { |a| ["APPIUM_SERVER_OUTPUT",
 "APPIUM_SERVER_LOG_OUTPUT"].include?(a.type) }
raise "No Appium server log artifacts found (expected APPIUM_SERVER_OUTPUT or
 APPIUM_SERVER_LOG_OUTPUT)." if filtered.empty?

# Prefer OUTPUT; else LOG_OUTPUT
chosen = filtered.find { |a| a.type == "APPIUM_SERVER_OUTPUT" } ||
         filtered.find { |a| a.type == "APPIUM_SERVER_LOG_OUTPUT" }

url = chosen.url
ext = (chosen.extension && !chosen.extension.empty?) ? chosen.extension : "log"
out_path = "./appium_server_log.#{ext}"

# 2) Download the artifact
uri = URI.parse(url)
Net::HTTP.start(uri.host, uri.port, use_ssl: (uri.scheme == "https")) do |http|
  req = Net::HTTP::Get.new(uri)
  http.request(req) do |resp|
    raise "Failed GET: #{resp.code} #{resp.body}" unless resp.code.to_i / 100 == 2
```

```
    File.open(out_path, "wb") { |f| resp.read_body { |chunk| f.write(chunk) } }
  end
end
puts "Saved Appium server log to: #{File.expand_path(out_path)}"
```

Isso mostrará uma saída como a seguinte:

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', allowInsecure: [ 'execute_driver_script', ... ],
 useDrivers: [ 'uiautomator' ] }
```

Capacidades e comandos compatíveis do Appium

O endpoint Appium do Device Farm suporta a maioria dos mesmos comandos e recursos desejados que você usa em dispositivos locais, com algumas exceções. As listas a seguir mostram quais recursos e comandos não são suportados atualmente. Se seus testes não puderem ser executados conforme o esperado devido a uma capacidade restrita, abra um caso de suporte para obter orientação adicional.

Recursos com suporte

Ao criar uma sessão do Appium no Device Farm, recomendamos ter um conjunto distinto de recursos que exclua quaisquer recursos específicos do seu dispositivo local. No Device Farm, a criação da sessão pode falhar se determinados recursos não suportados forem definidos. Isso inclui recursos específicos do dispositivo, como e. `udid` `platformVersion` Além disso, alguns recursos relacionados ao ChromeDriver WebDriverAgent Android e ao iOS não são compatíveis, bem como recursos que só são compatíveis com emuladores e simuladores.

Comandos compatíveis

A maioria dos comandos do Appium que são executados corretamente em dispositivos Android e iOS reais serão executados conforme o esperado no Device Farm, com as seguintes exclusões:

Comandos do dispositivo Appium () **`/appium/device`**

- `install_app`
- `finger_print`

- send_sms
- gsm_call
- gsm_signal
- gsm_voice
- power_ac
- power_capacity
- network_speed
- shake

Métodos e scripts de execução do Appium () **/execute**

- installApp
- execEmuConsoleCommand
- fingerprint
- gsmCall
- gsmSignal
- sendSms
- gsmVoice
- powerAC
- powerCapacity
- networkSpeed
- sensorSet
- injectEmulatorCameraImage
- isGpsEnabled
- shake
- clearApp
- clearKeychains
- configureLocalization
- enrollBiometric
- getPasteboard
- installXCTestBundle

- listXCTestBundles
- listXCTestsInTestBundle
- runXCTest
- sendBiometricMatch
- setPasteboard
- setPermission
- startAudioRecording
- startLogsBroadcast
- startRecordingScreen
- startScreenStreaming
- startXCTestScreenRecording
- stopAudioRecording
- stopLogsBroadcast
- stopRecordingScreen
- stopScreenStreaming
- stopXCTestScreenRecording
- updateSafariPreferences

Dispositivos privados no AWS Device Farm

Um dispositivo privado é um dispositivo móvel físico que o AWS Device Farm implementa em seu nome em um data center da Amazon. Este dispositivo é exclusivo para sua AWS conta.

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

Se tiver uma frota de dispositivos privados, você pode criar sessões de acesso remoto e programar execuções de teste usando seus dispositivos privados. Para obter mais informações, consulte Criar uma execução de teste ou iniciar uma sessão de acesso remoto no AWS Device Farm. Você também pode criar perfis de instância para controlar o comportamento dos seus dispositivos privados durante uma sessão de acesso remoto ou a execução de um teste. Para obter mais informações, consulte Criar um perfil de instância no AWS Device Farm. Opcionalmente, você pode solicitar que determinados dispositivos Android privados sejam implantados como dispositivos rooteados.

Você também pode criar um serviço de endpoint de nuvem privada virtual da Amazon para testar aplicações privadas às quais sua empresa tem acesso, mas que não podem ser acessados pela Internet. Por exemplo, você pode ter um aplicativo web em execução dentro da sua VPC que você deseja testar em dispositivos móveis. Para obter mais informações, consulte Usar os serviços de endpoint da Amazon VPC com o Device Farm: legado (não recomendado).

Se você tiver interesse em usar uma frota de dispositivos privados, fale conosco. A equipe do Device Farm deve trabalhar com você para configurar e implantar uma frota de dispositivos privados AWS em sua conta.

Tópicos

- Criar um perfil de instância no AWS Device Farm
- Solicitar dispositivos privados adicionais no AWS Device Farm
- Criar uma execução de teste ou iniciar uma sessão de acesso remoto no AWS Device Farm
- Selecionar dispositivos privados em um grupo de dispositivos no AWS Device Farm
- Ignorar a nova assinatura de aplicações em dispositivos privados no AWS Device Farm
- Amazon VPC em todas AWS as regiões no AWS Device Farm

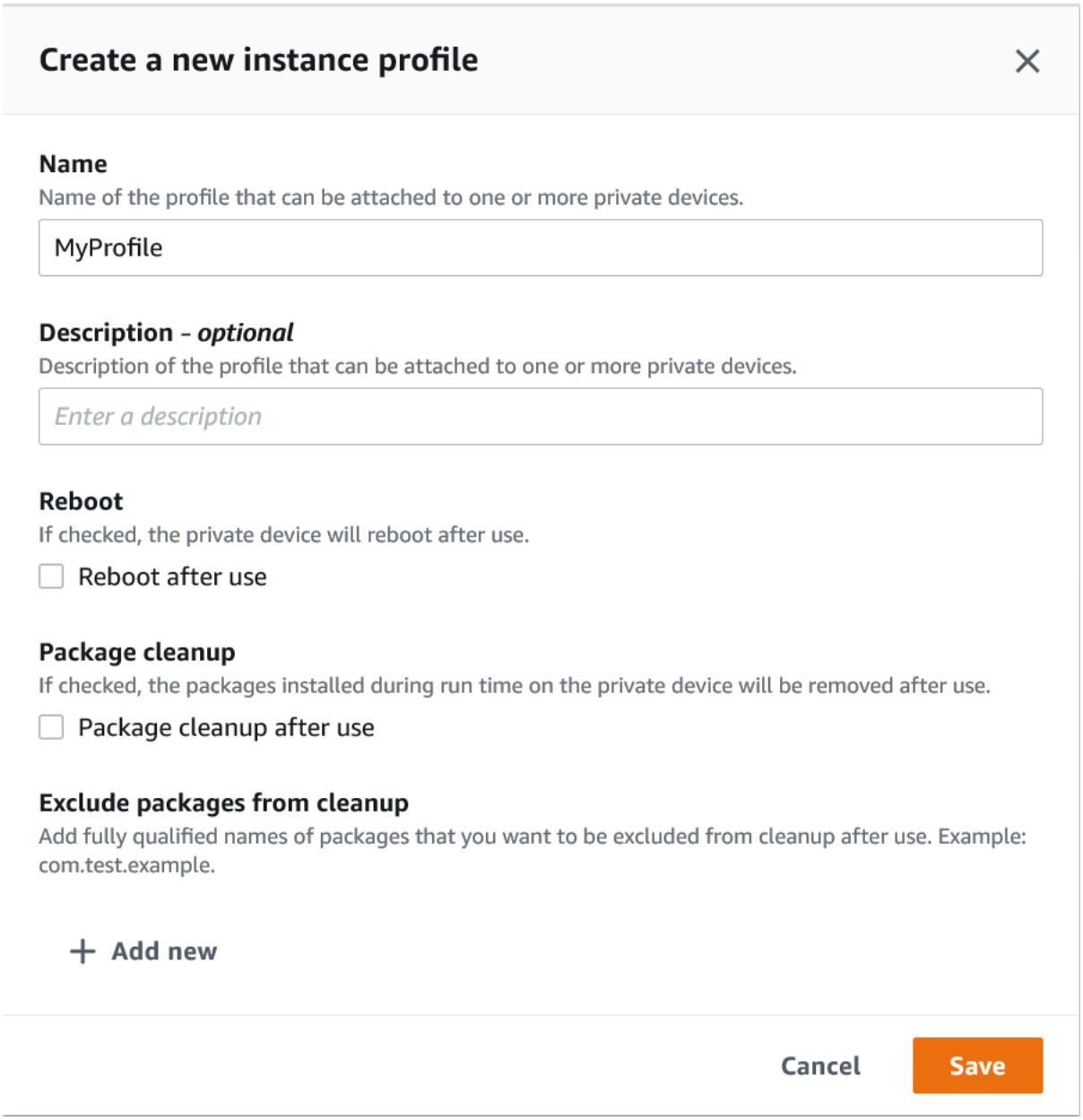
- Encerrar dispositivos privados no Device Farm

Criar um perfil de instância no AWS Device Farm

Você pode configurar uma frota que contém um ou mais dispositivos privados. Esses dispositivos são dedicados à sua conta do AWS . Depois de configurar os dispositivos, você pode opcionalmente criar um ou mais perfis de instância para eles. Perfis de instância podem ajudar você a automatizar execuções de teste e consistentemente aplicar as mesmas configurações para instâncias do dispositivo. Os perfis de instâncias também podem ajudar você a controlar o comportamento da sessão de acesso remoto. Para acessar mais informações sobre dispositivos privados no Device Farm, consulte Dispositivos privados no AWS Device Farm.

Para criar uma instância

1. Abra o console do Device Farm em https://console.aws.amazon.com/devicefarm/.
2. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e Dispositivos privados.
3. Escolha Perfis de instância.
4. Escolha Criar perfil de instância.
5. Insira um nome para o perfil de instância.



6. (Opcional) Digite uma descrição do perfil de instância.
7. (Opcional) Altere qualquer uma das seguintes configurações para especificar quais ações você deseja que o Device Farm execute em um dispositivo após o término de cada execução de teste ou sessão:

   - Reinicia após o uso: para reiniciar o dispositivo, marque essa caixa de seleção. Por padrão, essa caixa de seleção fica desmarcada (`false`).
   - Limpeza do pacote: para remover todos os pacotes de aplicações que você instalou no dispositivo, marque essa caixa de seleção. Por padrão, essa caixa de seleção fica

desmarcada (`false`). Para manter todos os pacotes de aplicativos que você instalou no dispositivo, deixe esta caixa de seleção desmarcada.

- Excluir pacotes da limpeza: para manter apenas pacotes selecionados de aplicações no dispositivo, marque a caixa de seleção Limpeza do pacote e escolha Adicionar novo. Para o nome do pacote, insira o nome totalmente qualificado do pacote de aplicativos que você deseja manter no dispositivo (por exemplo, `com.test.example`). Para manter mais pacotes de aplicações no dispositivo, escolha Adicionar novo e, em seguida, digite o nome totalmente qualificado de cada pacote.

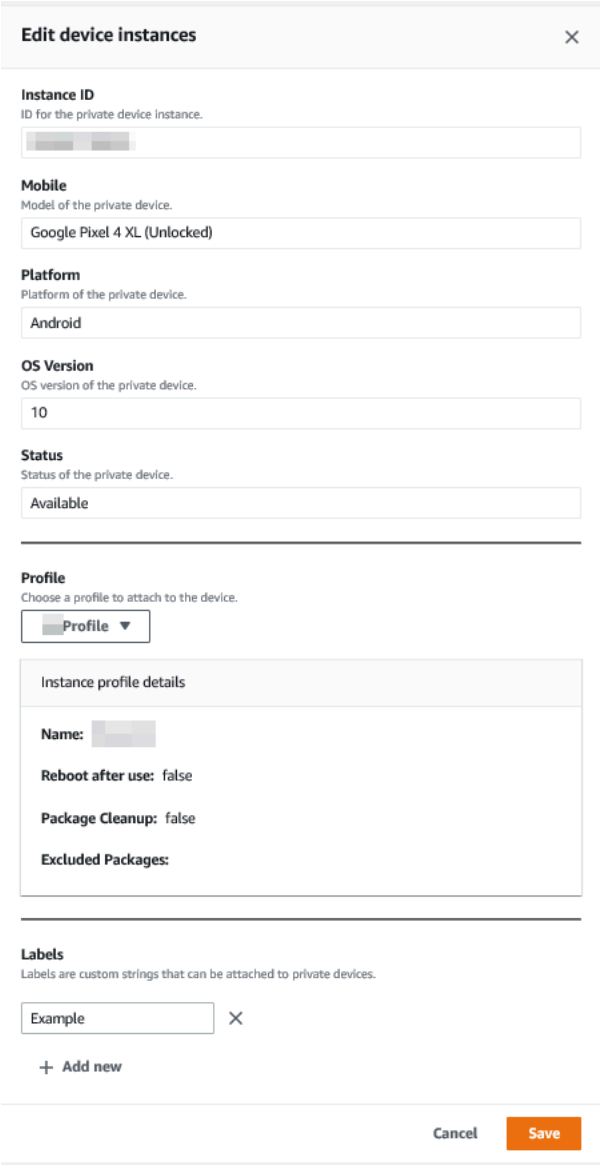
8. Escolha Salvar.

Solicitar dispositivos privados adicionais no AWS Device Farm

No AWS Device Farm, é possível solicitar que instâncias adicionais de dispositivos privados sejam adicionadas à sua frota. Você também pode visualizar e alterar as configurações das instâncias de dispositivos privados existentes em sua frota. Para acessar mais informações sobre dispositivos privados, consulte Dispositivos privados no AWS Device Farm.

Como solicitar dispositivos privados adicionais ou alterar suas configurações

1. Abra o console do Device Farm em https://console.aws.amazon.com/devicefarm/.
2. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e Dispositivos privados.
3. Escolha Instâncias do dispositivo. A guia Instâncias do dispositivo exibe uma tabela dos dispositivos privados em sua frota. Para pesquisar ou filtrar rapidamente a tabela, insira os termos de pesquisa na barra de pesquisa acima das colunas.
4. Para solicitar uma nova instância de dispositivo privado, escolha Solicitar a instância do dispositivo ou fale conosco. Os dispositivos privados exigem configuração adicional com a ajuda da equipe do Device Farm.
5. Na tabela de instâncias de dispositivos, selecione a opção de alternância ao lado da instância sobre a qual você deseja visualizar informações ou gerenciar e selecione Editar.



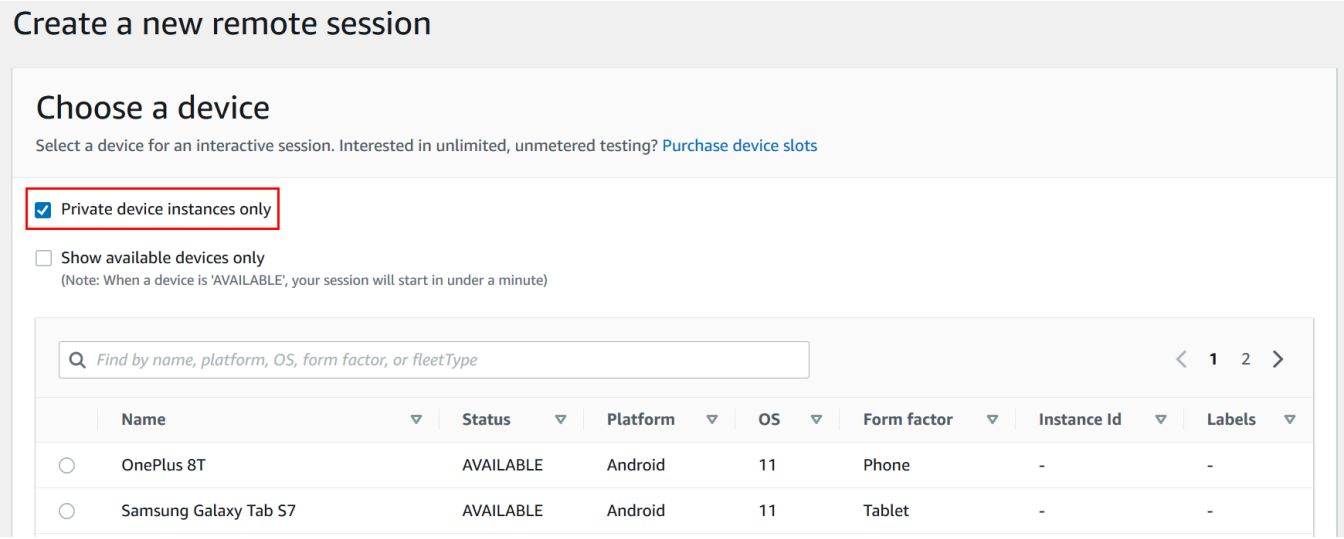
6. Para anexar um perfil de instância à instância do dispositivo, escolha-a na lista suspensa Perfil. Anexar um perfil de instância pode ser útil se você deseja sempre excluir um pacote de aplicação específico das tarefas de limpeza, por exemplo. Para acessar mais informações sobre como usar perfis de instância com dispositivos, consulte Criar um perfil de instância no AWS Device Farm.
7. (Opcional) Em Rótulos, escolha Adicionar novo para adicionar um rótulo à instância do dispositivo. Os rótulos podem ajudar você a categorizar seus dispositivos e encontrar dispositivos específicos com mais facilidade.
8. Escolha Salvar.

Criar uma execução de teste ou iniciar uma sessão de acesso remoto no AWS Device Farm

No AWS Device Farm, depois de configurar uma frota de dispositivos privados, você pode criar execuções de teste ou iniciar sessões de acesso remoto com um ou mais dispositivos privados em sua frota. Para acessar mais informações sobre dispositivos privados, consulte Dispositivos privados no AWS Device Farm.

Como criar uma execução de teste ou uma sessão de acesso remoto

1. Abra o console do Device Farm em https://console.aws.amazon.com/devicefarm/.
2. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e, em seguida, Projetos.
3. Escolha um projeto existente na lista ou crie um novo. Para criar um projeto, selecione Novo projeto, insira um nome para o projeto e selecione Enviar.
4. Execute um destes procedimentos:
   - Para criar uma execução de teste, escolha Testes automatizados e, em seguida, Criar nova execução. O assistente orienta você durante as etapas para criação da execução. Na etapa Selecionar dispositivos, você pode editar um pool de dispositivos existente ou criar um novo pool de dispositivos que inclua somente os dispositivos privados que a equipe do Device Farm configurou e associou à sua AWS conta. Para obter mais informações, consulte the section called “Criar um grupo de dispositivos privados”.
   - Para iniciar uma sessão de acesso remoto, escolha Acesso remoro, em seguida, Iniciar nova sessão. Na página Escolha um dispositivo, selecione Somente instâncias privadas de dispositivos para limitar a lista somente aos dispositivos privados que a equipe do Device Farm configurou e associou à sua AWS conta. Em seguida, escolha o dispositivo que você deseja acessar, insira um nome para a sessão de acesso remoto e escolha Confirmar e iniciar a sessão.



Selecionar dispositivos privados em um grupo de dispositivos no AWS Device Farm

Para usar dispositivos privados em sua execução de teste, você pode criar um pool de dispositivos que seleciona seus dispositivos privados. Os pools de dispositivos permitem que você selecione dispositivos privados principalmente por meio de três tipos de regras de pool de dispositivos:

1. Regras com base no ARN do dispositivo
2. Regras com base no rótulo da instância do dispositivo
3. Regras com base no ARN da instância do dispositivo

Nas seções a seguir, cada tipo de regra e seus casos de uso são descritos detalhadamente. Você pode usar o console Device Farm, a Interface de Linha de AWS Comando (AWS CLI) ou a API Device Farm para criar ou modificar um pool de dispositivos com dispositivos privados usando essas regras.

Tópicos

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

ARN do dispositivo

O ARN de um dispositivo é um identificador que representa um tipo de dispositivo em vez de qualquer instância específica de dispositivo físico. Um tipo de dispositivo é definido pelos seguintes atributos:

- O ID da frota do dispositivo
- O OEM do dispositivo
- O número do modelo do dispositivo
- A versão do sistema operacional do dispositivo.
- O estado do dispositivo que indica se ele está enraizado ou não

Muitas instâncias de dispositivos físicos podem ser representadas por um único tipo de dispositivo, em que cada instância desse tipo tem os mesmos valores para esses atributos. Por exemplo, se você tivesse três `Apple iPhone 13` dispositivos na versão iOS `16.1.0` em sua frota privada, cada dispositivo compartilharia o mesmo ARN do dispositivo. Se algum dispositivo fosse adicionado ou removido da sua frota com esses mesmos atributos, o ARN do dispositivo continuaria a representar quaisquer dispositivos disponíveis que você tivesse em sua frota para esse tipo de dispositivo.

O ARN do dispositivo é a maneira mais robusta de selecionar dispositivos privados para um pool de dispositivos, pois permite que o pool de dispositivos continue selecionando dispositivos, independentemente das instâncias de dispositivos específicas que você implantou a qualquer momento. Instâncias individuais de dispositivos privados podem apresentar falhas de hardware, fazendo com que o Device Farm as substitua automaticamente por novas instâncias funcionais do mesmo tipo de dispositivo. Nesses cenários, a regra de ARN do dispositivo garante que seu pool de dispositivos possa continuar selecionando dispositivos no caso de uma falha de hardware.

Quando você usa uma regra de ARN de dispositivo para dispositivos privados em seu pool de dispositivos e agenda uma execução de teste com esse pool, o Device Farm verifica automaticamente quais instâncias de dispositivos privados são representadas pelo ARN desse dispositivo. Das instâncias que estão disponíveis atualmente, uma delas será designada para executar seu teste. Se nenhuma instância estiver disponível no momento, o Device Farm aguardará até que a primeira instância disponível do ARN desse dispositivo fique disponível e a atribuirá para executar seu teste.

Rótulos de instância do dispositivo

Um rótulo de instância de dispositivo é um identificador textual que você pode anexar como metadados para uma instância de dispositivo. Você pode anexar vários rótulos a cada instância do dispositivo e o mesmo rótulo a várias instâncias do dispositivo. Para obter mais informações sobre como adicionar, modificar ou remover rótulos de dispositivos de instâncias de dispositivos, consulte Managing private devices.

O rótulo da instância do dispositivo pode ser uma forma robusta de selecionar dispositivos privados para um pool de dispositivos porque, se você tiver várias instâncias de dispositivos com o mesmo rótulo, ele permitirá que o pool de dispositivos selecione qualquer uma delas para seu teste. Se o ARN do dispositivo não for uma boa regra para seu caso de uso (por exemplo, se você quiser selecionar entre dispositivos de vários tipos de dispositivos ou se quiser selecionar entre um subconjunto de todos os dispositivos de um tipo de dispositivo), os rótulos de instância do dispositivo poderão permitir que você selecione entre vários dispositivos para seu pool de dispositivos com maior granularidade. Instâncias individuais de dispositivos privados podem apresentar falhas de hardware, fazendo com que o Device Farm as substitua automaticamente por novas instâncias funcionais do mesmo tipo de dispositivo. Nesses cenários, a instância do dispositivo substituto não reterá nenhum metadado do rótulo da instância do dispositivo substituído. Portanto, se você aplicar o mesmo rótulo de instância de dispositivo a várias instâncias de dispositivos, a regra de rótulo de instância de dispositivo garantirá que seu pool de dispositivos possa continuar selecionando instâncias de dispositivos no caso de uma falha de hardware.

Quando você usa uma regra de rótulo de instância de dispositivo para dispositivos privados em seu pool de dispositivos e agenda uma execução de teste com esse pool, o Device Farm verifica automaticamente quais instâncias de dispositivos particulares são representadas por esse rótulo de instância de dispositivo e, dessas instâncias, seleciona aleatoriamente uma que esteja disponível para executar seu teste. Se nenhuma estiver disponível, o Device Farm selecionará aleatoriamente qualquer instância de dispositivo com o rótulo de instância do dispositivo para executar seu teste e colocará o teste em fila para execução no dispositivo quando estiver disponível.

Instância ARN

O ARN de uma instância de dispositivo é um identificador que representa uma instância física de dispositivo bare metal implantada em uma frota privada. Por exemplo, se você tivesse três *iPhone 13* dispositivos no sistema operacional *15.0.0* em sua frota privada, enquanto cada dispositivo compartilhasse o mesmo ARN do dispositivo, cada dispositivo também teria seu próprio ARN de instância representando apenas essa instância.

O ARN da instância do dispositivo é a maneira menos eficiente de selecionar dispositivos privados para um pool de dispositivos e só é recomendado se os rótulos do dispositivo ARNs e da instância do dispositivo não se adequarem ao seu caso de uso. As instâncias de dispositivos geralmente ARNs são usadas como regras para grupos de dispositivos quando uma instância de dispositivo específica é configurada de forma única e específica como um pré-requisito para seu teste e se essa configuração precisa ser conhecida e verificada antes que o teste seja executado nela. Instâncias individuais de dispositivos privados podem apresentar falhas de hardware, fazendo com que o Device Farm as substitua automaticamente por novas instâncias funcionais do mesmo tipo de dispositivo. Nesses cenários, a instância do dispositivo substituto terá um ARN de instância de dispositivo diferente do dispositivo substituído. Portanto, se você depende da instância de dispositivos ARNs para seu pool de dispositivos, precisará alterar manualmente a definição da regra do pool de dispositivos de usar o ARN antigo para usar o novo ARN. Se você precisar pré-configurar manualmente o dispositivo para o teste, esse pode ser um fluxo de trabalho eficaz (comparado ao dispositivo ARNs). Para testes em grande escala, é recomendável tentar adaptar esses casos de uso para trabalhar com rótulos de instância de dispositivos e, se possível, ter várias instâncias de dispositivos pré-configuradas para testes.

Quando você usa uma regra ARN de instância de dispositivo para dispositivos privados em seu pool de dispositivos e agenda uma execução de teste com esse pool, o Device Farm atribui automaticamente esse teste a essa instância de dispositivo. Se essa instância do dispositivo não estiver disponível, o Device Farm colocará o teste em fila no dispositivo assim que ele estiver disponível.

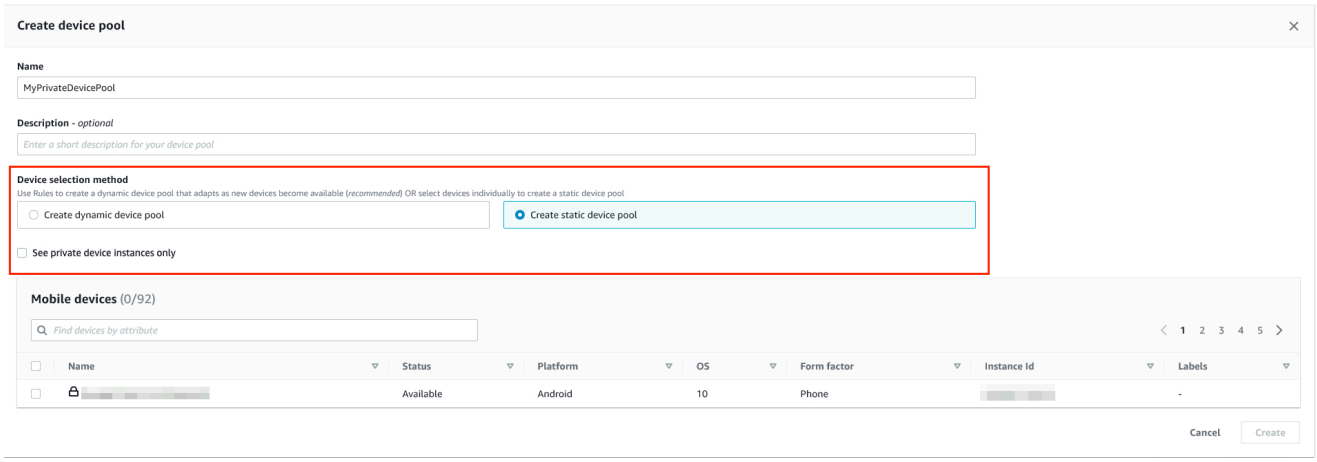
Criação de um pool de dispositivos privados com dispositivos privados (console)

Ao criar uma execução de teste, você pode criar um grupo de dispositivos para a execução de teste e garantir que o grupo inclua apenas os dispositivos privados.

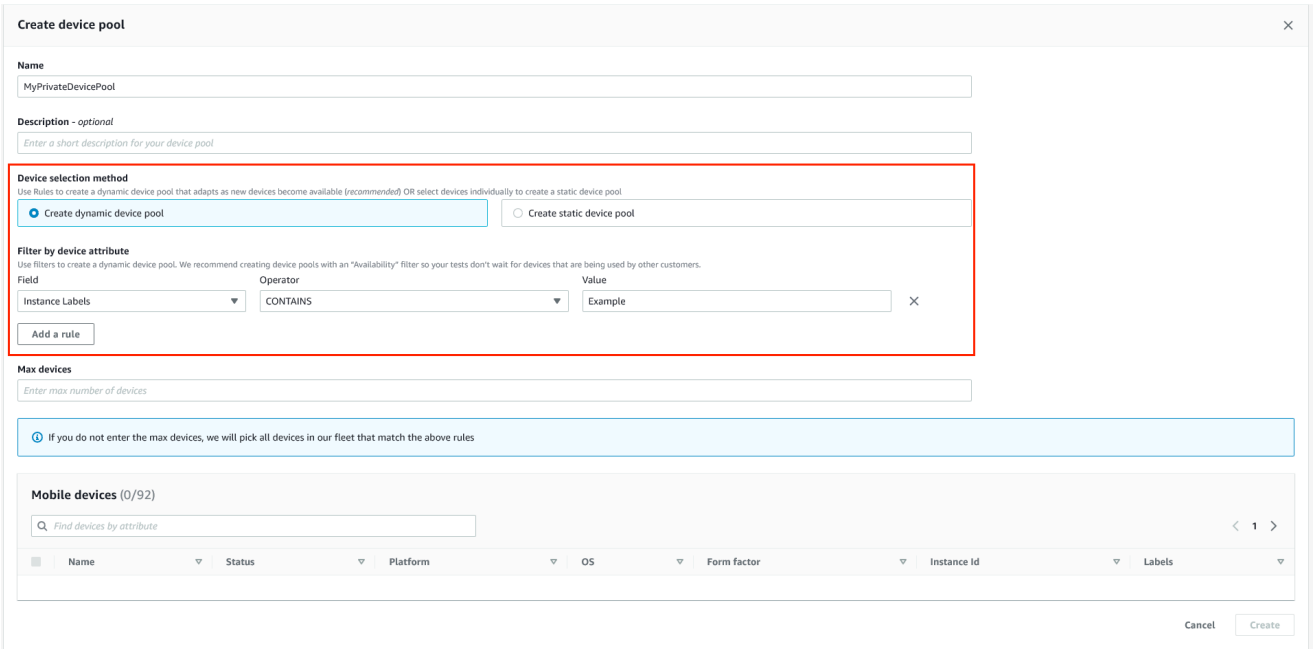
> **Note**
>
> Ao criar um pool de dispositivos com dispositivos privados no console, você só pode usar qualquer uma das três regras disponíveis para selecionar dispositivos privados. Se você quiser criar um pool de dispositivos que contenha vários tipos de regras para dispositivos privados (por exemplo, pools de dispositivos que contêm regras para dispositivos ARNs e instâncias de dispositivos ARNs), é necessário criar o pool por meio da CLI ou da API.

1. Abra o console do Device Farm em https://console.aws.amazon.com/devicefarm/.
2. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e, em seguida, Projetos.
3. Escolha um projeto existente na lista ou crie um novo. Para criar um projeto, selecione Novo projeto, insira um nome para o projeto e selecione Enviar.
4. Escolha Configurações do projeto e, depois, navegue até a guia Grupos de dispositivos.
5. Escolha Criar grupo de dispositivos e insira um nome e uma descrição opcional para esse grupo.

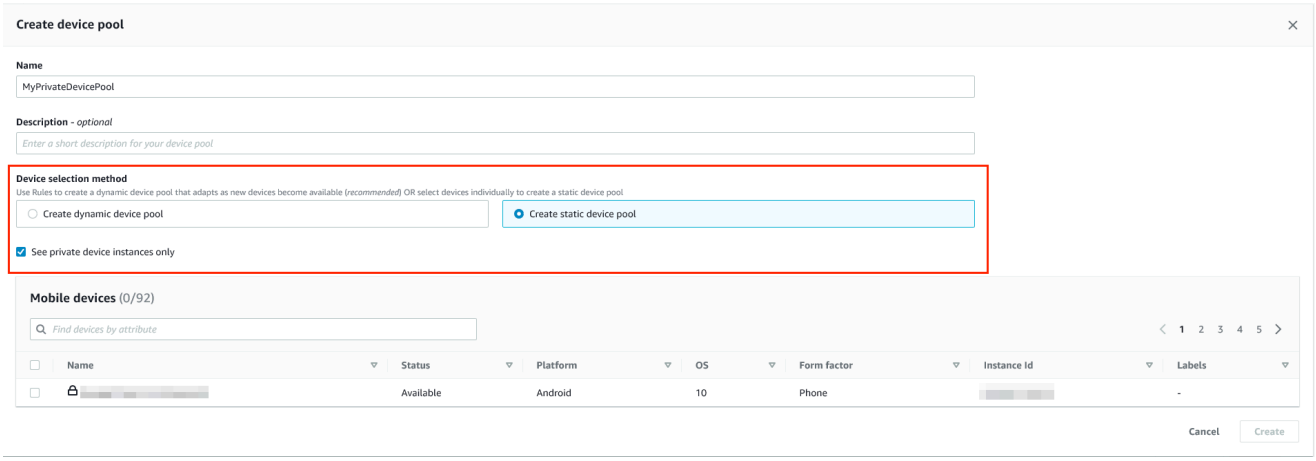
    a. Para usar regras de ARN de dispositivos para seu grupo de dispositivos, escolha Criar grupos estáticos de dispositivos e selecione os tipos de dispositivos específicos da lista que você gostaria de usar no grupo de dispositivos. Não selecione Somente instâncias de dispositivos privados porque essa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



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



c. Para usar regras de ARN de instância de dispositivo para seu pool de dispositivos, escolha Create static device pool e selecione Private device instances only para limitar a lista de dispositivos somente às instâncias privadas que o Device Farm associou à sua AWS conta.



6. Escolha Criar.

Criação de um grupo de dispositivos privados com dispositivos privados (AWS CLI)

- Execute o comando create-device-pool.

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

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

- Chame a API `CreateDevicePool`.

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

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

A assinatura de aplicações é um processo que envolve a assinatura digital de um pacote de aplicações (por exemplo, APK, IPA) com uma chave privada para que ele possa ser instalado em um dispositivo ou publicado em uma loja de aplicações, como a Google Play Store ou a Apple App Store. Para simplificar os testes reduzindo o número de assinaturas e perfis necessários e aumentar a segurança dos dados em dispositivos remotos, o AWS Device Farm assinará novamente sua aplicação após o upload para o serviço.

Depois de fazer upload da sua aplicação para o AWS Device Farm, o serviço gerará uma nova assinatura para ela usando seus próprios certificados de assinatura e perfis de provisionamento. Esse processo substitui a assinatura original da aplicação pela assinatura do AWS Device Farm. A aplicação assinada novamente é então instalada nos dispositivos de teste fornecidos pelo AWS Device Farm. A nova assinatura possibilita que a aplicação seja instalada e executada nesses dispositivos sem a necessidade dos certificados originais do desenvolvedor.

No iOS, substituímos o perfil de aprovisionamento incorporado por um perfil curinga e assinamos novamente o aplicativo. Se você os fornecer, adicionaremos dados auxiliares ao pacote da aplicação antes da instalação para que os dados estejam presentes na respectiva sandbox. Assinar novamente o aplicativo iOS resulta na remoção de todos os direitos.

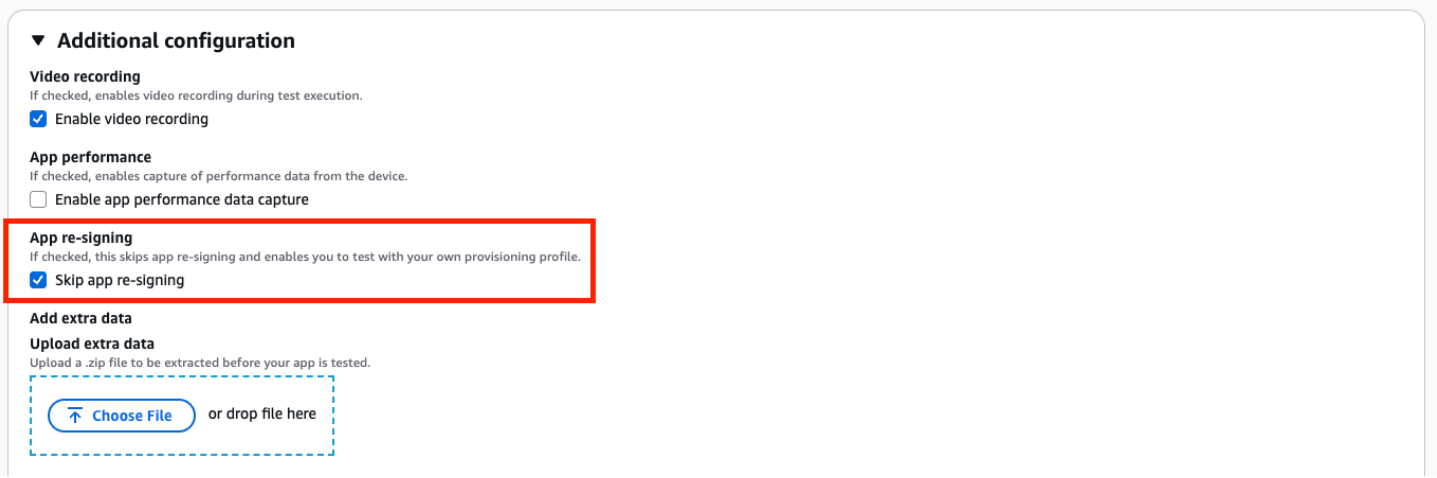
No Android, assinamos novamente o aplicativo. Isso pode interromper a funcionalidade que depende da assinatura da aplicação, como a API do Android do Google Maps. Também pode acionar a detecção antipirataria e antiadulteração disponível em produtos como. DexGuard Em relação a

testes integrados, podemos modificar o manifesto a fim de incluir as permissões necessárias para capturar e salvar capturas de tela.

Quando você usa dispositivos privados, pode ignorar a etapa em que o AWS Device Farm assina novamente a aplicação. Isso é diferente dos dispositivos públicos, em que o Device Farm sempre assina novamente a aplicação nas plataformas Android e iOS.

Você pode ignorar a nova assinatura do aplicativo ao criar uma sessão de acesso remoto ou a execução de um teste. Isso poderá ser útil se a aplicação tiver uma funcionalidade que seja interrompida quando o Device Farm assinar novamente a aplicação. Por exemplo, notificações por push podem não funcionar depois da nova assinatura. Para obter mais informações sobre as alterações que o Device Farm faz ao testar seu aplicativo, consulte o AWS Device Farm FAQs ou a página Apps.

Para ignorar a nova assinatura da aplicação para um teste, selecione Ignorar a nova assinatura da aplicação em Configuração adicional. Essa opção só está disponível em dispositivos privados.



> **Note**
>
> Se você estiver usando a XCTest estrutura, a opção Ignorar a reassinatura do aplicativo não estará disponível. Para obter mais informações, consulte Integrando o Device Farm com XCTest o iOS.

Etapas adicionais para configurar suas configurações de assinatura de aplicativos variam, dependendo de você estar usando dispositivos privados Android ou iOS.

Ignorar a nova assinatura da aplicação em dispositivos Android

Se você estiver testando o aplicativo em um dispositivo Android privado, selecione Skip app re-signing (Ignorar nova assinatura do aplicativo) ao criar a execução de teste ou a sessão de acesso remoto. Nenhuma outra configuração é necessária.

Ignorar a nova assinatura da aplicação em dispositivos iOS

A Apple exige que você assine um aplicativo para teste antes de carregá-lo em um dispositivo. Para dispositivos iOS, você tem duas opções para assinar seu aplicativo.

- Se estiver usando um perfil de desenvolvedor interno (Enterprise), você pode passar para a próxima seção, the section called “Criar uma sessão de acesso remoto para confiar na sua aplicação”.

- Se você estiver usando um perfil de desenvolvimento de aplicativo iOS ad hoc, você deverá primeiramente registrar o dispositivo com a conta de desenvolvedor da Apple e atualizar o perfil de provisionamento para incluir o dispositivo privado. Você deve assinar novamente o aplicativo com o perfil de provisionamento que você atualizou. Você pode executar a aplicação assinada novamente no Device Farm.

Para registrar um dispositivo com um perfil de aprovisionamento de desenvolvimento de aplicativos iOS ad hoc

1. Faça login em sua conta de desenvolvedor da Apple.
2. Navegue até a seção IDsCertificados e Perfis do console.
3. Vá até Devices (Dispositivos).
4. Registre o dispositivo em sua conta de desenvolvedor da Apple. Para obter o nome e o UDID do dispositivo, use a operação `ListDeviceInstances` da API do Device Farm.
5. Acesse seu perfil de provisionamento e escolha Editar.
6. Selecione o dispositivo na lista.
7. No XCode, busque seu perfil de provisionamento atualizado e, em seguida, assine novamente o aplicativo.

Nenhuma outra configuração é necessária. Agora, você pode criar uma sessão de acesso remoto ou uma execução de teste e selecionar Skip app re-signing (Ignorar nova assinatura do aplicativo).

Criar uma sessão de acesso remoto para confiar na sua aplicação para iOS

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

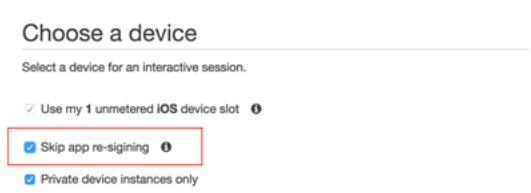
Para fazer isso, você deve instalar um aplicativo de espaço reservado assinado com o mesmo certificado do aplicativo que você deseja testar. Depois que o dispositivo confia no perfil de configuração ou no desenvolvedor do aplicativo corporativo, todos os aplicativos desse desenvolvedor são confiáveis no dispositivo privado até que você os exclua. Portanto, ao instalar novas versões do aplicativo que deseja testar, você não precisará confiar no desenvolvedor do aplicativo novamente a cada vez. 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.

Um procedimento comum que muitos clientes usam é assinar novamente o aplicativo de amostra Device Farm para iOS e instalá-lo em seus dispositivos como aplicativo de espaço reservado.

Antes de iniciar sua sessão de acesso remoto, siga as etapas em Criar um perfil de instância no AWS Device Farm para criar ou modificar um perfil de instância no Device Farm. No perfil da instância, adicione o ID do pacote do aplicativo de espaço reservado à configuração Excluir pacotes da limpeza. Em seguida, anexe o perfil de instância à instância do dispositivo privado para garantir que o Device Farm não remova essa aplicação do dispositivo antes de iniciar uma nova execução de teste. Isso garante que o certificado do desenvolvedor permaneça confiável.

Você pode carregar o aplicativo de espaço reservado para o dispositivo usando uma sessão de acesso remoto, que permite iniciar o aplicativo e confiar no desenvolvedor.

1. Siga as instruções em Criar uma sessão para criar uma sessão de acesso remoto usando o perfil da instância de dispositivo privado que você acabou de criar. Ao criar a sessão, não se esqueça de selecionar Skip app re-signing (Ignorar nova assinatura do aplicativo).



> **Important**
>
> Para filtrar a lista de dispositivos de modo a incluir apenas dispositivos privados, selecione Private device instances only (Somente instâncias de dispositivos privados) para garantir que você use um dispositivo privado com o perfil de instância correto.

Certifique-se também de adicionar o aplicativo de espaço reservado ou o aplicativo que você deseja testar à configuração Excluir pacotes da limpeza do perfil de instância anexado a essa instância.

2. Quando a sessão remota for iniciada, escolha Escolher arquivo para instalar uma aplicação que use seu perfil de provisionamento interno.
3. Inicie o aplicativo recém-carregado.
4. Confirme se uma caixa de diálogo do iOS aparece indicando que o desenvolvedor do aplicativo corporativo não é confiável.
5. Em seguida, se o dispositivo iOS estiver na versão 18 ou superior, abra um ticket de suporte com a equipe do AWS Device Farm para que nossa equipe confie no aplicativo para você, pois esses dispositivos exigem que o aplicativo seja manualmente confiável. Caso contrário, se a versão do iOS for 17 ou inferior, você poderá acessar o aplicativo Configurações e, em Configurações gerais, confiar no aplicativo no menu VPN e Perfis.

Todos os aplicativos desse perfil de configuração ou do desenvolvedor do aplicativo empresarial já são confiáveis nesse dispositivo privado até que você os exclua.

Amazon VPC em todas AWS as regiões no AWS Device Farm

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

Há duas maneiras de acessar seus serviços privados localizados em uma região diferente. Se você tiver serviços localizados em uma outra região que não seja `us-west-2`, poderá usar o emparelhamento de VPC para emparelhar a VPC dessa região com outra VPC que esteja fazendo

interface com o Device Farm em `us-west-2`. No entanto, se você tiver serviços em várias regiões, um Transit Gateway permitirá que você acesse esses serviços com uma configuração de rede mais simples.

Para obter mais informações, consulte VPC peering scenarios no Guia de emparelhamento da Amazon VPC.

Visão geral do emparelhamento de VPC VPCs em diferentes regiões no AWS Device Farm

Você pode emparelhar quaisquer dois VPCs em regiões diferentes, desde que tenham blocos CIDR distintos e não sobrepostos. Isso garante que todos os endereços IP privados sejam exclusivos e permite que todos os recursos do se dirijam uns VPCs aos outros sem a necessidade de qualquer forma de tradução de endereços de rede (NAT). Para obter mais informações sobre notação de CIDR, consulte RFC 4632.

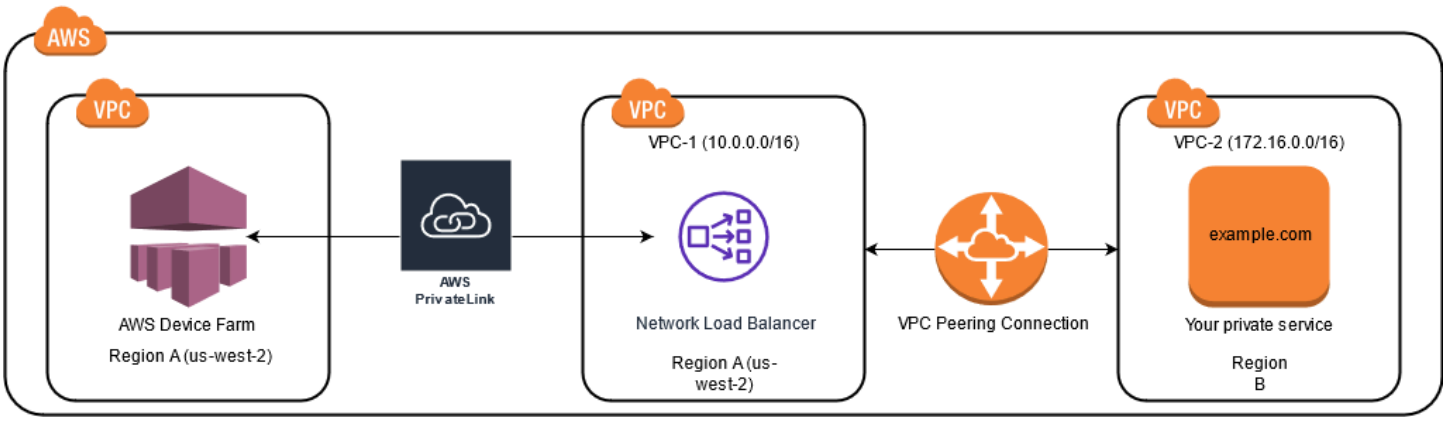
Este tópico inclui um cenário de exemplo entre regiões no qual o Device Farm (denominado VPC-1) está localizado na região Oeste dos EUA (Oregon) (`us-west-2`). A segunda VPC neste exemplo (chamada de VPC-2) está em outra região.

Exemplo de VPC entre regiões do Device Farm

Componente da VPC	VPC-1	VPC-2
CIDR	10.0.0.0/16	172.16.0.0/16

> **Important**
>
> Estabelecer uma conexão de peering entre dois VPCs pode mudar a postura de segurança do. VPCs Além disso, adicionar novas entradas às tabelas de rotas pode alterar a postura de segurança dos recursos dentro do VPCs. É sua responsabilidade implementar essas configurações de forma a atender aos requisitos de segurança de sua organização. Para obter mais informações, consulte o Modelo de responsabilidade compartilhada.

O diagrama a seguir mostra os componentes do exemplo e as interações entre esses componentes.



Tópicos

- Pré-requisitos para usar o Amazon VPC no AWS Device Farm
- Etapa 1: configurar uma conexão de emparelhamento entre a VPC-1 e a VPC-2
- Etapa 2: atualizar as tabelas de rotas na VPC-1 e na VPC-2
- Etapa 3: Criar um grupo de destino
- Etapa 4: Criando um Network Load Balancer
- Etapa 5: criar um serviço de endpoint da VPC para conectar sua VPC ao Device Farm
- Etapa 6: criar uma configuração do endpoint da VPC entre a VPC e o Device Farm
- Etapa 7: criar uma execução de teste para usar a configuração do endpoint da VPC
- Criar uma rede escalável com o Transit Gateway

Pré-requisitos para usar o Amazon VPC no AWS Device Farm

Este exemplo requer o seguinte:

- Dois VPCs que são configurados com sub-redes contendo blocos CIDR não sobrepostos.
- A VPC-1 deve estar na região `us-west-2` e conter sub-redes para as zonas de disponibilidade `us-west-2a`, `us-west-2b` e `us-west-2c`.

Para obter mais informações sobre como criar VPCs e configurar sub-redes, consulte Trabalho com sub-redes VPCs e sub-redes no Amazon VPC Peering Guide.

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

Estabeleça uma conexão de emparelhamento entre os dois VPCs contendo blocos CIDR não sobrepostos. Para fazer isso, consulte Create and accept VPC peering connections no Guia de emparelhamento da Amazon VPC. Usando o cenário entre regiões deste tópico e o Guia de emparelhamento da Amazon VPC, é criado o seguinte exemplo de configuração de conexão de emparelhamento:

Nome

```
Device-Farm-Peering-Connection-1
```

ID VPC (solicitante)

```
vpc-0987654321gfedcba (VPC-2)
```

Conta

```
My account
```

Região

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

ID VPC (Aceitante)

```
vpc-1234567890abcdefg (VPC-1)
```

> **Note**
>
> Certifique-se de consultar as cotas de conexão de emparelhamento da VPC ao estabelecer novas conexões de emparelhamento. Para obter mais informações, consulte Amazon VPC quotas no Guia de emparelhamento da Amazon VPC.

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

Depois de configurar uma conexão de peering, você deve estabelecer uma rota de destino entre os dois VPCs para transferir dados entre eles. Para estabelecer essa rota, você pode atualizar manualmente a tabela de rotas da VPC-1 para apontar para a sub-rede da VPC-2 e vice-versa.

Para fazer isso, consulte Update your route tables for a VPC peering connection no Guia de emparelhamento da Amazon VPC. Usando o cenário entre regiões deste tópico e o Guia de emparelhamento da Amazon VPC, é criado o seguinte exemplo de configuração de tabela de rotas:

Exemplo de tabela de rotas VPC do Device Farm

Componente da VPC	VPC-1	VPC-2
ID da tabela de rotas	rtb-1234567890abcdefg	rtb-0987654321gfedcba
Faixa de endereços locais	10.0.0.0/16	172.16.0.0/16
Intervalo de endereços de destino	172.16.0.0/16	10.0.0.0/16

Etapa 3: Criar um grupo de destino

Depois de definir as rotas de destino, você pode configurar um Network Load Balancer na VPC-1 para rotear as solicitações para a VPC-2.

O Network Load Balancer deve primeiro conter um grupo de destino que contenha os endereços IP para os quais as solicitações são enviadas.

Para criar um grupo de destino

1. Identifique os endereços IP do serviço que você deseja segmentar no VPC-2.
   - Esses endereços IP devem ser membros da sub-rede usada na conexão de emparelhamento.
   - Os endereços IP direcionados devem ser estáticos e imutáveis. Se o seu serviço tiver endereços IP dinâmicos, considere a possibilidade de direcionar um recurso estático (como um Network Load Balancer) e fazer com que esse recurso estático encaminhe as solicitações para o seu verdadeiro destino.

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

Load Balancer. 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, você pode abrir o console do Amazon EC2 em e https://console.aws.amazon.com/ec2/, em seguida, escolher Interfaces de rede. A partir daí, você pode visualizar os endereços IP de cada uma das interfaces de rede do Network Load Balancer em cada zona de disponibilidade.

2. Crie um grupo de destino na VPC-1. Para fazer isso, consulte Criar um grupo-alvo para o seu Network Load Balancer no Guia do Usuário para Network Load Balancers.

   Os grupos de destino para serviços em uma VPC diferente exigem a seguinte configuração:

   - Em Escolher um tipo de destino, escolha Endereços IP.
   - Para VPC, escolha a VPC que hospedará o balanceador de carga. Para o exemplo deste tópico, essa será a VPC-1.
   - Na página Registrar alvos, registre um alvo para cada endereço IP na VPC-2.

     Em Rede, escolha Outro endereço IP privado.

     Em Zona de disponibilidade, escolha as zonas desejadas na VPC-1.

     Para IPv4 endereço, escolha o endereço IP VPC-2.

     Em Portas, escolha suas portas.
   - Escolha Incluir como pendente abaixo. Quando terminar de especificar os endereços, selecione Registrar alvos pendentes.

Usando o cenário entre regiões deste tópico e o Guia do usuário de Network Load Balancers, os seguintes valores são usados na configuração do grupo de destino:

Target type

```
IP addresses
```

Nome do grupo-alvo

```
my-target-group
```

Protocolo/porta

```
TCP : 80
```

VPC

```
vpc-1234567890abcdefg (VPC-1)
```

Rede

```
Other private IP address
```

Zona de disponibilidade

```
all
```

IPv4 endereço

```
172.16.100.60
```

Portas

```
80
```

Etapa 4: Criando um Network Load Balancer

Crie um Network Load Balancer usando o grupo de destino descrito na etapa 3. Para fazer isso, consulte Criação de um Network Load Balancer.

Usando o cenário entre regiões deste tópico, os seguintes valores são usados em um exemplo de configuração do Network Load Balancer:

Nome do load balancer

```
my-nlb
```

Esquema

```
Internal
```

VPC

```
vpc-1234567890abcdefg (VPC-1)
```

Mapeamento

```
us-west-2a - subnet-4i23iuufkdiufsloi
```

```
us-west-2b - subnet-7x989pkjj78nmn23j
```

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

Protocolo/porta

```
TCP : 80
```

Grupo de destino

```
my-target-group
```

Etapa 5: criar um serviço de endpoint da VPC para conectar sua VPC ao Device Farm

Você pode usar o Network Load Balancer para criar um serviço de endpoint da VPC. Por meio desse serviço de endpoint da VPC, o Device Farm pode se conectar ao seu serviço na VPC-2 sem nenhuma infraestrutura adicional, como um gateway da internet, uma instância NAT ou uma conexão VPN.

Para fazer isso, consulte Creating an Amazon VPC endpoint service.

Etapa 6: criar uma configuração do endpoint da VPC entre a VPC e o Device Farm

Agora você pode estabelecer uma conexão privada entre sua VPC e o Device Farm. Você pode usar o Device Farm para testar serviços privados sem expô-los à Internet pública. Para fazer isso, consulte Creating a VPC endpoint configuration in Device Farm.

Usando o cenário entre regiões deste tópico, os seguintes valores são usados em um exemplo de configuração de endpoint de VPC:

Nome

```
My VPCE Configuration
```

Nome do serviço VPCE

```
com.amazonaws.vpce.us-west-2.vpce-svc-1234567890abcdefg
```

Nome DNS do serviço

```
devicefarm.com
```

Etapa 7: criar uma execução de teste para usar a configuração do endpoint da VPC

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

Criar uma rede escalável com o Transit Gateway

Para criar uma rede escalável usando mais de duas VPCs, você pode usar o Transit Gateway para atuar como um hub de trânsito de rede para interconectar sua rede VPCs e a rede local. Para configurar uma VPC na mesma região que o Device Farm para usar um Transit Gateway, você pode seguir o guia Amazon VPC endpoint services with Device Farm para direcionar recursos em outra região com base em seus endereços IP privados.

Para obter mais informações sobre o Transit Gateway, consulte What is a transit gateway? no Guia de Transit Gateways da Amazon VPC.

Encerrar dispositivos privados no Device Farm

`<Para encerrar um dispositivo privado após o prazo inicial acordado, você deve f`
Para acessar mais informações sobre dispositivos privados, consulte Dispositivos privados no AWS Device Farm.

> **Important**
>
> Essas instruções se aplicam somente à rescisão de contratos de dispositivos privados. Para todos os outros AWS serviços e problemas de cobrança, consulte a respectiva documentação desses produtos ou entre em contato com o AWS suporte.

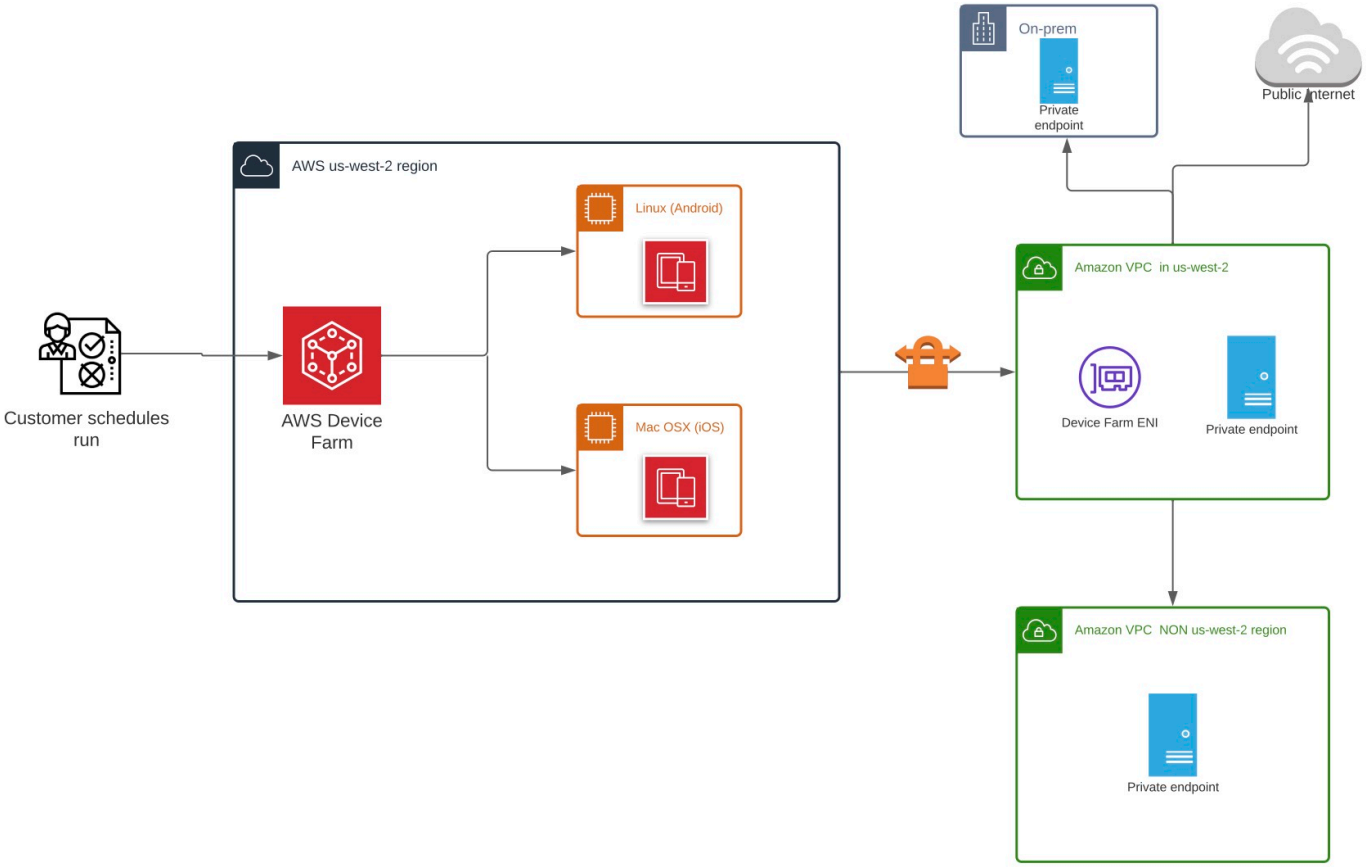
VPC-ENI no AWS Device Farm

> **Warning**
>
> Esse recurso só está disponível em dispositivos privados. Para solicitar o uso privado de um dispositivo em sua AWS conta, entre em contato conosco. Se você já tiver dispositivos privados adicionados à sua AWS conta, é altamente recomendável usar esse método de conectividade VPC.

O recurso de conectividade VPC-ENI do AWS Device Farm ajuda os clientes a se conectarem com segurança a seus endpoints privados hospedados no software local ou em outro provedor de AWS nuvem.

Você pode conectar os dispositivos móveis do Device Farm e suas máquinas host a um ambiente Amazon Virtual Private Cloud (Amazon VPC) na `us-west-2` região, o que permite o acesso a non-internet-facing serviços e aplicativos isolados por meio de uma interface de rede elástica. Para obter mais informações sobre VPCs, consulte o Guia do usuário da Amazon VPC.

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



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

> **Note**
>
> O Device Farm não cobra nada pela criação ENIs na VPC de um cliente em. `us-west-2` O custo da conectividade entre regiões ou entre VPCs externas não está incluído nesse recurso.

Depois de configurar o acesso à VPC, os dispositivos e as máquinas host que você usa para seus testes não conseguirão se conectar a recursos fora da VPC (por exemplo, públicos CDNs), a menos

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

Tópicos

AWS controle de acesso e IAM

O AWS Device Farm permite que você use o AWS Identity and Access Management (IAM) para criar políticas que concedem ou restringem o acesso aos recursos do Device Farm. Para usar o recurso de conectividade VPC com o AWS Device Farm, a seguinte política do IAM é necessária para a conta de usuário ou função que você está usando para acessar o AWS Device Farm:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [{
      "Effect": "Allow",
      "Action": [
        "devicefarm:*",
        "ec2:DescribeVpcs",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:CreateNetworkInterface"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
```

```
      "Resource": "arn:aws:iam::*:role/aws-service-role/devicefarm.amazonaws.com/
AWSServiceRoleForDeviceFarm",
      "Condition": {
        "StringLike": {
          "iam:AWSServiceName": "devicefarm.amazonaws.com"
        }
      }
    }
  ]
}
```

Para criar ou atualizar um projeto Device Farm com uma configuração de VPC, sua política de IAM deve permitir que você chame as seguintes ações em relação aos recursos listados na configuração da VPC:

```
"ec2:DescribeVpcs"
"ec2:DescribeSubnets"
"ec2:DescribeSecurityGroups"
"ec2:CreateNetworkInterface"
```

Além disso, sua política do IAM também deve permitir a criação do perfil vinculado ao serviço:

```
"iam:CreateServiceLinkedRole"
```

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

Perfis vinculados ao serviço

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

Um perfil vinculado ao serviço facilita a configuração do Device Farm porque você não precisa adicionar manualmente as permissões necessárias. O Device Farm define as permissões de suas

funções vinculadas ao serviço e, a menos que seja definido de outra forma, somente o Device Farm pode assumir suas funções. As permissões definidas incluem a política de confiança e a política de permissões, que não pode ser anexada a nenhuma outra entidade do IAM.

Um perfil vinculado ao serviço poderá ser excluído somente após excluir seus atributos relacionados. Isso protege os recursos do Device Farm porque você não pode remover inadvertidamente a permissão para acessar os recursos.

Para obter informações sobre outros serviços que oferecem suporte aos perfis vinculados ao serviço, consulte Serviços da AWS que funcionam com o IAM e procure os serviços com Sim na coluna Perfil vinculado ao serviço. Escolha um Sim com um link para exibir a documentação da função vinculada a serviço desse serviço.

Permissões de perfil vinculado ao serviço para o Device Farm

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

A função AWSService RoleForDeviceFarm vinculada ao serviço confia nos seguintes serviços para assumir a função:

- `devicefarm.amazonaws.com`

A política de permissões de função permite que o Device Farm conclua as seguintes ações:

- Para sua conta
  - Criar interfaces de rede
  - Describe network interfaces
  - Descreva VPCs
  - Descrever sub-redes
  - Descrever grupos de segurança
  - Excluir interfaces
  - Modificar interfaces de rede
- Para interfaces de rede
  - Criar tags
- Para interfaces de rede do EC2 gerenciadas pela Device Farm

- Criar permissões de interface de rede

A política do IAM completa diz:

JSON

```
{
"Version":"2012-10-17",
"Statement": [
 {
  "Effect": "Allow",
  "Action": [
   "ec2:DescribeNetworkInterfaces",
   "ec2:DescribeVpcs",
   "ec2:DescribeSubnets",
   "ec2:DescribeSecurityGroups"
  ],
  "Resource": "*"
 },
 {
  "Effect": "Allow",
  "Action": [
   "ec2:CreateNetworkInterface"
  ],
  "Resource": [
   "arn:aws:ec2:*:*:subnet/*",
   "arn:aws:ec2:*:*:security-group/*"
  ]
 },
 {
  "Effect": "Allow",
  "Action": [
   "ec2:CreateNetworkInterface"
  ],
  "Resource": [
   "arn:aws:ec2:*:*:network-interface/*"
  ],
  "Condition": {
   "StringEquals": {
    "aws:RequestTag/AWSDeviceFarmManaged": "true"
   }
  }
```

```
},
{
 "Effect": "Allow",
 "Action": [
  "ec2:CreateTags"
 ],
 "Resource": "arn:aws:ec2:*:*:network-interface/*",
 "Condition": {
  "StringEquals": {
   "ec2:CreateAction": "CreateNetworkInterface"
  }
 }
},
{
 "Effect": "Allow",
 "Action": [
  "ec2:CreateNetworkInterfacePermission",
  "ec2:DeleteNetworkInterface"
 ],
 "Resource": "arn:aws:ec2:*:*:network-interface/*",
 "Condition": {
  "StringEquals": {
   "aws:ResourceTag/AWSDeviceFarmManaged": "true"
  }
 }
},
{
 "Effect": "Allow",
 "Action": [
  "ec2:ModifyNetworkInterfaceAttribute"
 ],
 "Resource": [
  "arn:aws:ec2:*:*:security-group/*",
  "arn:aws:ec2:*:*:instance/*"
 ]
},
{
 "Effect": "Allow",
 "Action": [
  "ec2:ModifyNetworkInterfaceAttribute"
 ],
 "Resource": "arn:aws:ec2:*:*:network-interface/*",
 "Condition": {
  "StringEquals": {
```

```
      "aws:ResourceTag/AWSDeviceFarmManaged": "true"
     }
    }
   }
  ]
}
```

Você deve configurar permissões para que uma entidade do IAM (por exemplo, um usuário, grupo ou função) crie, edite ou exclua um perfil vinculado a serviço. Para saber mais, consulte Permissões de Função Vinculadas ao Serviço no Guia do Usuário do IAM.

Criar um perfil vinculado ao serviço para o Device Farm

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

Se excluir esse perfil vinculado ao serviço e precisar criá-lo novamente, será possível usar esse mesmo processo para recriar o perfil em sua conta. Quando você cria seu primeiro recurso do Device Farm, o Device Farm cria o perfil vinculado ao serviço para você novamente.

Você também pode usar o console do IAM para criar um perfil vinculado ao serviço com o caso de uso do Device Farm. Na AWS CLI ou na AWS API, crie uma função vinculada ao serviço com o nome do `devicefarm.amazonaws.com` serviço. Para obter mais informações, consulte Criar uma função vinculada ao serviço no Guia do usuário do IAM. Se você excluir essa função vinculada ao serviço, será possível usar esse mesmo processo para criar a função novamente.

Editar um perfil vinculado ao serviço para o Device Farm

O Device Farm não permite que você edite a função AWSService RoleForDeviceFarm vinculada ao serviço. Depois que criar um perfil vinculado ao serviço, você não poderá alterar o nome do perfil, pois várias entidades podem fazer referência a ele. No entanto, será possível editar a descrição do perfil usando o IAM. Para saber mais, consulte Editar uma função vinculada a serviço no Guia do usuário do IAM.

Excluir um perfil vinculado ao serviço para o Device Farm

Se você não precisar mais usar um recurso ou serviço que requer um perfil vinculado ao serviço, é recomendável excluí-lo. Dessa forma, você não tem uma entidade não utilizada que não seja monitorada ativamente ou mantida. No entanto, você deve limpar os recursos de seu perfil vinculado ao serviço antes de excluí-lo manualmente.

> **Note**
>
> Se o serviço Device Farm estiver usando a função quando você tentar excluir os recursos, a exclusão poderá falhar. Se isso acontecer, espere alguns minutos e tente a operação novamente.

Como excluir manualmente o perfil vinculado ao serviço usando o IAM

Use o console do IAM AWS CLI, o ou a AWS API para excluir a função AWSService RoleForDeviceFarm vinculada ao serviço. Para saber mais, consulte Excluir um perfil vinculado ao serviço no Guia do usuário do IAM.

Regiões compatíveis com as funções vinculadas ao serviço Device Farm

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

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

Nome da região	Identidade da região	Support em Device Farm
Leste dos EUA (Norte da Virgínia)	us-east-1	Não
Leste dos EUA (Ohio)	us-east-2	Não
Oeste dos EUA (N. da Califórnia)	us-west-1	Não
Oeste dos EUA (Oregon)	us-west-2	Sim
Ásia-Pacífico (Mumbai)	ap-south-1	Não

Nome da região	Identidade da região	Support em Device Farm
Ásia-Pacífico (Osaka)	ap-northeast-3	Não
Ásia-Pacífico (Seul)	ap-northeast-2	Não
Ásia-Pacífico (Singapura)	ap-southeast-1	Não
Ásia-Pacífico (Sydney)	ap-southeast-2	Não
Ásia-Pacífico (Tóquio)	ap-northeast-1	Não
Canadá (Central)	ca-central-1	Não
Europa (Frankfurt)	eu-central-1	Não
Europa (Irlanda)	eu-west-1	Não
Europa (Londres)	eu-west-2	Não
Europa (Paris)	eu-west-3	Não
América do Sul (São Paulo)	sa-east-1	Não
AWS GovCloud (US)	us-gov-west-1	Não

Pré-requisitos

A lista a seguir descreve alguns requisitos e sugestões a serem analisados ao criar configurações de VPC-ENI:

- Dispositivos privados devem ser atribuídos à sua AWS conta.
- Você deve ter uma AWS conta, usuário ou função com permissões para criar uma função vinculada ao serviço. Ao usar endpoints da Amazon VPC com os recursos de teste móvel do Device Farm, o Device Farm cria uma função vinculada ao AWS Identity and Access Management serviço (IAM).
- O Device Farm pode se conectar VPCs somente na `us-west-2` região. Se você não tiver uma VPC na região `us-west-2`, precisará criar uma. A seguir, para acessar recursos em uma VPC em outra região, você deve estabelecer uma conexão de emparelhamento entre a VPC na região `us-`

`west-2` e a VPC na outra região. Para obter informações sobre peering VPCs, consulte o Amazon VPC Peering Guide.

Você deve verificar se tem acesso à sua VPC especificada ao configurar a conexão. Você deve configurar determinadas permissões do Amazon Elastic Compute Cloud (Amazon EC2) do Device Farm.

- A resolução de DNS é necessária na VPC que você usa.
- Depois que a VPC for criada, você precisará das seguintes informações sobre a VPC na região `us-west-2`:
  - ID da VPC
  - Sub-rede IDs (somente sub-redes privadas)
  - Grupo de segurança IDs
- Você deve configurar as conexões da Amazon VPC por projeto. No momento, você pode configurar somente uma configuração de VPC por projeto. Quando você configura uma VPC, a Amazon VPC cria uma interface dentro da sua VPC e a atribui às sub-redes e grupos de segurança especificados. Todas as sessões futuras associadas ao projeto usarão a conexão VPC configurada.
- Você não pode usar as configurações do VPC-ENI junto com o recurso VPCE legado.
- É altamente recomendável não atualizar um projeto existente com uma configuração de VPC-ENI, pois os projetos existentes podem ter configurações de VPCE que persistem no nível de execução. Em vez disso, se você já usa os recursos existentes do VPCE, use o VPC-ENI para todos os novos projetos.

Conectando o Amazon VPC

Você pode configurar e atualizar seu projeto para usar endpoints da Amazon VPC. A configuração da VPC-ENI é configurada para cada projeto. Um projeto só pode ter um endpoint da VPC-ENI por vez. Para configurar o acesso da VPC a um projeto, você deve conhecer os seguintes detalhes:

- O ID da VPC em `us-west-2` se a aplicação estiver hospedada lá, ou o ID da VPC de `us-west-2` que se conecta a outra VPC em uma região diferente.
- Os grupos de segurança aplicáveis a serem aplicados à conexão.
- As sub-redes que serão associadas à conexão. Quando uma sessão é iniciada, a maior sub-rede disponível é usada. Recomendamos ter várias sub-redes associadas a diferentes zonas de disponibilidade para melhorar a postura de disponibilidade da sua conectividade de VPC.

- Ao usar o VPC-ENI, o resolvedor de DNS usado pelos hosts e dispositivos de teste do Device Farm será o servidor fornecido pelos serviços DHCP na sub-rede do cliente. Em uma configuração padrão, esse será o resolvedor padrão da VPC. Os clientes que desejam especificar resolvedores de DNS personalizados podem configurar um conjunto de opções DHCP em sua VPC.

Depois de criar sua configuração de VPC-ENI, você pode atualizar seus detalhes usando o console ou a CLI usando as etapas abaixo.

Console

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. No painel de navegação do Device Farm, escolha Teste para dispositivos móveis e, em seguida, Projetos.
3. Em Projetos de teste móvel, escolha o nome do seu projeto na lista.
4. Escolha Configurações do projeto.
5. Na seção Configurações da Virtual Private Cloud (VPC), você pode alterar a `VPC`, as `Subnets` (somente sub-redes privadas) e os `Security Groups`.
6. Escolha Salvar.

CLI

Use o seguinte comando AWS CLI para atualizar o Amazon VPC:

```
$  aws devicefarm update-project \
--arn arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef \
--vpc-config \
securityGroupIds=sg-02c1537701a7e3763,sg-005dadf9311efda25,\
subnetIds=subnet-09b1a45f9cac53717,subnet-09b1a45f9cac12345,\
vpcId=vpc-0238fb322af81a368
```

Você também pode configurar uma Amazon VPC ao criar seu projeto:

```
$  aws devicefarm create-project \
--name VPCDemo \
--vpc-config \
securityGroupIds=sg-02c1537701a7e3763,sg-005dadf9311efda25,\
subnetIds=subnet-09b1a45f9cac53717,subnet-09b1a45f9cac12345,\
```

```
vpcId=vpc-0238fb322af81a368
```

Limites

As seguintes limitações são aplicáveis ao recurso VPC-ENI:

- Você pode fornecer até cinco grupos de segurança na configuração de VPC de um projeto Device Farm.
- Você pode fornecer até oito sub-redes na configuração de VPC de um projeto Device Farm.
- Ao configurar um projeto do Device Farm para funcionar com sua VPC, a menor sub-rede que você pode fornecer deve ter no mínimo cinco endereços disponíveis. IPv4
- Não há suporte para endereços IP públicos no momento. Em vez disso, recomendamos usar sub-redes privadas em seus projetos do Device Farm. Se precisar de acesso público à Internet durante os testes, use um gateway de conversão de endereços de rede (NAT). Configurar um projeto do Device Farm com uma sub-rede pública não dá a seus testes acesso à Internet nem a um endereço IP público.
- A integração do VPC-ENI comporta somente sub-redes privadas em sua VPC.
- Somente o tráfego de saída da ENI gerenciada pelo serviço é permitido. Isso significa que a ENI não pode receber solicitações de entrada não solicitadas da VPC.

Usar os serviços de endpoint da Amazon VPC com o Device Farm: legado (não recomendado)

> **Warning**
>
> É altamente recomendável usar a conectividade VPC-ENI descrita nesta página para conectividade de endpoints privados, pois o VPCE agora é considerado um recurso legado. O VPC-ENI oferece maior flexibilidade, configurações mais simples, é mais econômico e requer bem menos esforço de manutenção quando comparado ao método de conectividade VPCE.

> **Note**
>
> O uso de serviços de endpoint da VPC da Amazon com o Device Farm só é permitido para clientes com dispositivos privados configurados. Para habilitar sua conta AWS para usar esse recurso com dispositivos privados, entre em contato conosco.

A Amazon Virtual Private Cloud (Amazon VPC) é um AWS serviço que você pode usar para lançar AWS recursos em uma rede virtual que você define. Com uma VPC, você tem controle sobre suas configurações de rede, como o intervalo de endereços IP, sub-redes, tabelas de roteamento e gateways de rede.

Se você usa a Amazon VPC para hospedar aplicativos privados na AWS região Oeste dos EUA (Oregon) (`us-west-2`), você pode estabelecer uma conexão privada entre sua VPC e o Device Farm. Com essa conexão, você pode usar o Device Farm para testar aplicações privadas sem expô-las à Internet pública. Para permitir que sua AWS conta use esse recurso com dispositivos privados, entre em contato conosco.

Para conectar um recurso em na VPC ao Device Farm, você pode usar o console da Amazon VPC para criar um serviço de endpoint da VPC. Esse serviço de endpoint permite que você forneça o recurso na VPC ao Device Farm por meio de um endpoint da VPC do Device Farm. O serviço de endpoint fornece conectividade confiável e escalável ao Device Farm sem exigir um gateway de Internet, uma instância de conversão de endereços de rede (NAT) ou uma conexão VPN. Para obter mais informações, consulte VPC endpoint services (AWS PrivateLink) no Guia.AWS PrivateLink

> **Important**
>
> O recurso de endpoint VPC do Device Farm ajuda você a conectar com segurança serviços internos privados em sua VPC à VPC pública do Device Farm usando conexões. AWS PrivateLink Embora a conexão seja segura e privada, essa segurança depende da proteção de suas credenciais da AWS . Se suas AWS credenciais forem comprometidas, um invasor poderá acessar ou expor seus dados de serviço para o mundo externo.

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

Antes de começar

As informações a seguir são para usuários da Amazon VPC na região oeste dos EUA (Oregon) (`us-west-2`), com uma sub-rede em cada uma das seguintes zonas de disponibilidade: us-west-2a, us-west-2b e us-west-2c.

O Device Farm tem requisitos adicionais para os serviços de endpoint da VPC com os quais você pode usá-lo. Quando você criar e configurar um serviço de endpoint da VPC para trabalhar com o Device Farm, escolha opções que atendam aos seguintes requisitos:

- As zonas de disponibilidade do serviço devem incluir us-west-2a, us-west-2b e us-west-2c. O Network Load Balancer associado a um serviço de endpoint da VPC determina as zonas de disponibilidade para esse serviço de endpoint da VPC. Se o serviço de endpoint da VPC não mostrar todas as três zonas de disponibilidade, você deverá recriar o Network Load Balancer para ativar essas três zonas e, em seguida, associar novamente o Network Load Balancer ao serviço de endpoint.
- As entidades principais permitidas para o serviço de endpoint devem incluir o nome do recurso da Amazon (ARN) do endpoint da VPC do Device Farm (ARN do serviço). Depois de criar o serviço de endpoint, adicione o ARN do serviço de endpoint da VPC do Device Farm à sua lista de permissões para dar permissão ao Device Farm para acessar seu serviço de endpoint da VPC. Para obter o ARN do serviço de endpoint da VPC do Device Farm, fale conosco.

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

A próxima seção explica como criar um serviço de endpoint da Amazon VPC que atenda a esses requisitos.

Etapa 1: Criação de um Network Load Balancer

A primeira etapa para estabelecer uma conexão privada entre sua VPC e o Device Farm é criar um Network Load Balancer para rotear as solicitações para um grupo de destino.

New console

Para criar um Network Load Balancer usando o novo console

1. Abra o console do Amazon Elastic Compute Cloud (Amazon EC2) em. https://console.aws.amazon.com/ec2/
2. No painel de navegação, em Balanceamento de carga, escolha Balanceadores de carga.
3. Selecione Criar um balanceador de carga.
4. Em Network Load Balancer, escolha Criar.
5. Na página Criar Network Load Balancer, em Configuração básica, faça o seguinte:
   a. Insira um Nome para o balanceador de carga.
   b. Em Esquema, escolha Interno.
6. Em Mapeamento de rede, faça o seguinte:
   a. Escolha a VPC do grupo de destino.
   b. Selecione os seguintes Mapeamentos:
      - `us-west-2a`
      - `us-west-2b`
      - `us-west-2c`
7. Em Receptores e roteamento, use as opções Protocolo e Porta para escolher o grupo de destino.

   > **Note**
   >
   > Por padrão, o balanceamento de carga da zona de disponibilidade cruzada está desativado.
   > Como o balanceador de carga usa as zonas de disponibilidade `us-west-2a`, `us-west-2b` e `us-west-2c`, ele exige que os destinos sejam registrados em cada uma delas ou, se você registrar destinos em menos de todas as três zonas, exige a ativação do balanceamento de carga entre zonas. Caso contrário, o balanceador de carga pode não funcionar como esperado.

8. Selecione Criar um balanceador de carga.

Old console

Para criar um Network Load Balancer usando o console antigo

1. Abra o console do Amazon Elastic Compute Cloud (Amazon EC2) em. https://console.aws.amazon.com/ec2/
2. No painel de navegação, em Balanceamento de carga, escolha Balanceadores de carga.
3. Selecione Criar um balanceador de carga.
4. Em Network Load Balancer, escolha Criar.
5. Na página Configurar balanceador de carga, em Configuração básica, faça o seguinte:
   a. Insira um Nome para o balanceador de carga.
   b. Em Esquema, escolha Interno.
6. Em Receptores, selecione o Protocolo e a Porta que o grupo de destino está usando.
7. Em Zonas de disponibilidade, faça o seguinte:
   a. Escolha a VPC do grupo de destino.
   b. Selecione as seguintes zonas de disponibilidade:
      - `us-west-2a`
      - `us-west-2b`
      - `us-west-2c`
   c. Escolha Próximo: defina as configurações de segurança.
8. (Opcional) Defina suas configurações de segurança e, em seguida, selecione Próximo: configure o roteamento.
9. Na página Configurar roteamento, faça o seguinte:
   a. Em Grupo-alvo, selecione Grupo-alvo existente.
   b. Para Nome, escolha seu grupo-alvo.
   c. Escolha Próximo: registrar alvos.
10. Na página Registrar alvos, revise seus alvos e escolha Próximo: revisão.

> **Note**
>
> Por padrão, o balanceamento de carga da zona de disponibilidade cruzada está desativado.

> Como o balanceador de carga usa as zonas de disponibilidade `us-west-2a`, `us-west-2b` e `us-west-2c`, ele exige que os destinos sejam registrados em cada uma delas ou, se você registrar destinos em menos de todas as três zonas, exige a ativação do balanceamento de carga entre zonas. Caso contrário, o balanceador de carga pode não funcionar como esperado.

11. Revise a configuração do balanceador de carga e escolha Criar.

Etapa 2: criar um serviço de endpoint da Amazon VPC

Depois de criar o Network Load Balancer, use o console da Amazon VPC para criar um serviço de endpoint na VPC.

1. Abra o console da Amazon VPC em https://console.aws.amazon.com/vpc/.
2. Em Recursos por região, escolha Serviços de endpoint.
3. Escolha Criar serviço de endpoint.
4. Execute um destes procedimentos:
   - Se você já tiver um Network Load Balancer que deseja que o serviço de endpoint use, escolha-o em Balanceadores de carga disponíveis e passe para a etapa 5.
   - Se você ainda não tiver criado um Network Load Balancer, escolha Criar novo balanceador de carga. O console do Amazon EC2 é aberto. Siga as etapas em Criar um Network Load Balancer, começando pela etapa 3, e continue com essas etapas no console da Amazon VPC.
5. Para as zonas de disponibilidade incluídas, verifique se `us-west-2a`, `us-west-2b` e `us-west-2c` aparecem na lista.
6. Se você não quiser aceitar ou negar manualmente cada solicitação de conexão enviada ao serviço de endpoint, em Configurações adicionais, desmarque Aceitação necessária. Se você desmarcar essa caixa de seleção, o serviço endpoint aceitará automaticamente cada solicitação de conexão que receber.
7. Escolha Criar.
8. No novo serviço de endpoint, escolha Permitir principais.
9. Fale conosco para obter o ARN do endpoint da VPC do Device Farm (ARN do serviço) a ser adicionado à lista de permissões do serviço de endpoint e, depois, adicione esse ARN do serviço à lista de permissões do serviço.

10. Na guia Detalhes do serviço endpoint, anote o nome do serviço - (nome do serviço). Você precisará desse nome ao criar a configuração do VPC endpoint na próxima etapa.

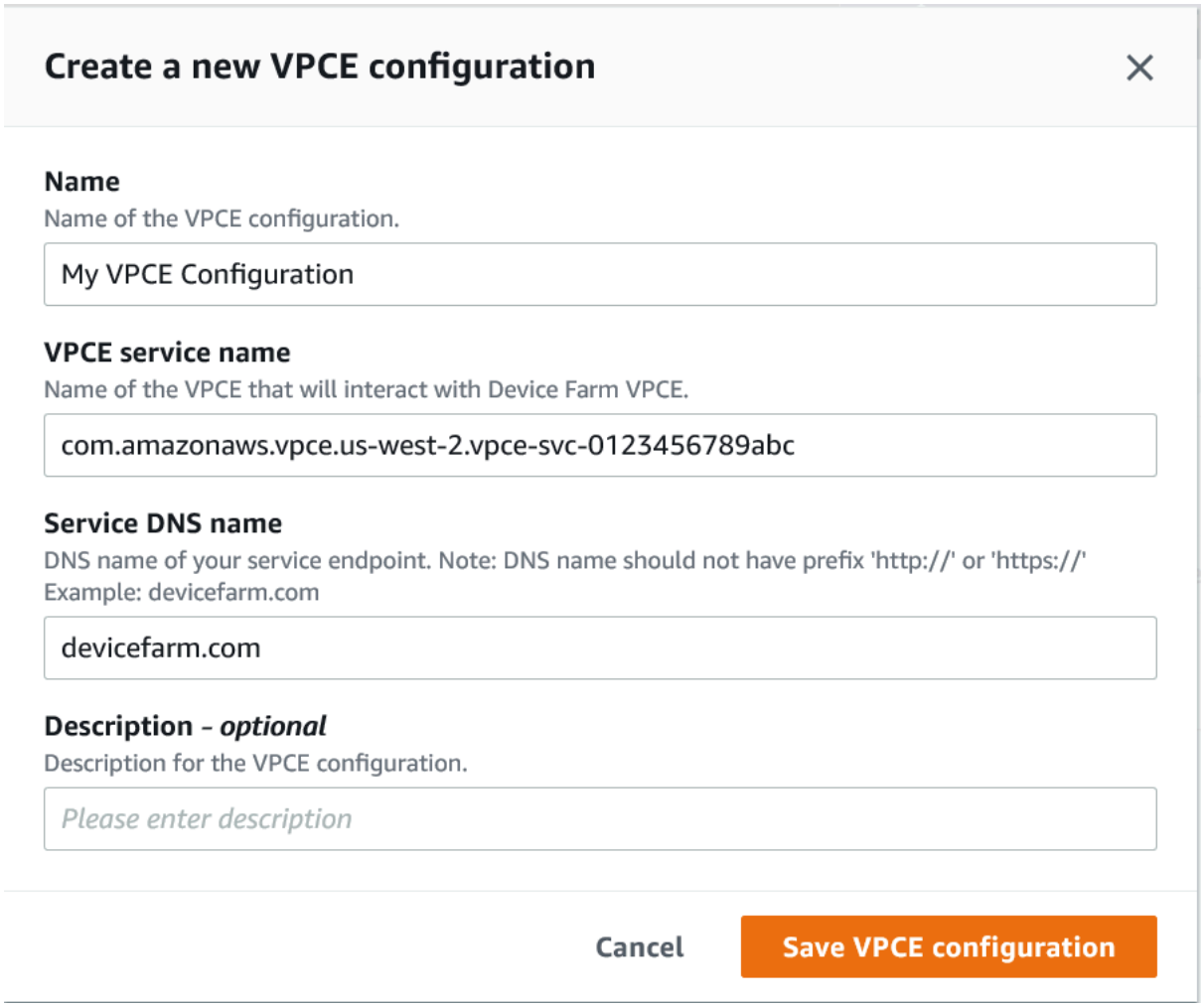
O serviço de endpoint da VPC agora está pronto para ser usado com o Device Farm.

Etapa 3: criar uma configuração de endpoint da VPC no Device Farm

Depois de criar um serviço de endpoint na Amazon VPC, você pode criar uma configuração de endpoint da Amazon VPC no Device Farm.

1. Faça login no console do Device Farm em https://console.aws.amazon.com/devicefarm.
2. No painel de navegação, escolha Teste para dispositivos móveis e Dispositivos privados.
3. Escolha Configurações de VPCE.
4. Escolha Criar uma configuração de VPCE.
5. Em Criar uma nova configuração de VPCE, insira um Nome para a configuração do endpoint da VPC.
6. Em Nome do serviço de VPCE, insira o nome do serviço de endpoint da Amazon VPC (nome do serviço) que você anotou no console da Amazon VPC. O nome é semelhante ao `com.amazonaws.vpce.us-west-2.vpce-svc-id`.
7. Em Nome DNS do serviço, digite o nome DNS do serviço para a aplicação que você deseja testar (por exemplo, `devicefarm.com`). Não especifique `http` ou `https` antes do nome DNS do serviço.

   O nome de domínio não é acessível pela internet pública. Além disso, esse novo nome de domínio, que mapeia para o serviço de endpoint da VPC, é gerado pelo Amazon Route 53 e está disponível exclusivamente para você na sua sessão do Device Farm.
8. Escolha Salvar.



Etapa 4: Criar uma execução de teste

Depois de salvar a configuração do endpoint da VPC, você pode usar a configuração para criar execuções de teste ou sessões de acesso remoto. Para acessar mais informações, consulte Criar uma execução de teste no Device Farm ou Criar uma sessão.

Registro em log de chamadas de API do AWS Device Farm com o AWS CloudTrail

O AWS Device Farm é integrado ao AWS CloudTrail, um serviço que fornece um registro das ações realizadas por um usuário, perfil 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 habilitar a entrega contínua 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 AWS CloudTrailGuia do usuário.

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, ela é registrada em um evento do CloudTrail junto com eventos de outros serviços da AWS em Histórico de eventos. É possível visualizar, pesquisar e baixar eventos recentes em sua AWS conta. 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 da 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 log no bucket do Amazon S3 que você especificou Além disso, é possível configurar outros AWS serviços para melhor analisar e agir de acordo com dados coletados do evento nos logs CloudTrail. Para obter mais informações, consulte:

- Visão Geral para Criar uma Trilha
- Serviços e integrações com suporte no CloudTrail
- Configurando Notificações Amazon SNS para CloudTrail

- Recebendo Arquivos de Log CloudTrail de Várias Regiões e Recebendo Arquivos de Log CloudTrail de Várias Contas

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

Todas as ações do Device Farm são registradas em log e documentadas em AWS CLIReferência do e Automatização do Device Farm. Por exemplo, as chamadas para criar um novo projeto ou executar no Device Farm geram entradas nos arquivos de log 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 usuário do AWS Identity and Access Management (IAM).
- Se a solicitação foi feita com credenciais de segurança temporárias de um perfil ou de um usuário federado.
- Se a solicitação foi feita por outro AWS serviço.

Para mais informações, consulte Elemento userIdentity CloudTrail.

Compreensão das entradas do arquivo de log do AWS Device Farm

Uma trilha é uma configuração que permite a entrega de eventos como arquivos de log a um bucket Amazon S3 especificado. Os arquivos de log CloudTrail contêm uma ou mais entradas de log. Um evento representa uma única solicitação de qualquer fonte, e inclui informações sobre a ação solicitada, data e hora da ação, parâmetros da solicitação e assim por diante. Os arquivos de log do CloudTrail não são um rastreamento de pilha ordenada de chamadas de API pública, portanto, não são exibidos em uma ordem específica.

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

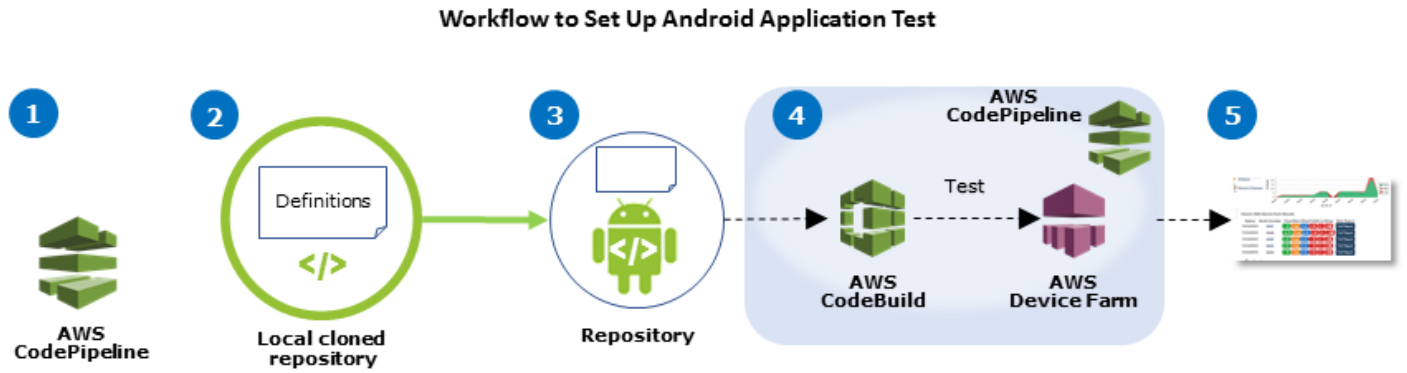
```
{
  "Records": [
    {
      "eventVersion": "1.03",
      "userIdentity": {
        "type": "Root",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:root",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext": {
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2015-07-08T21:13:35Z"
          }
        }
      },
      "eventTime":"2015-07-09T00:51:22Z",
      "eventSource": "devicefarm.amazonaws.com",
      "eventName":"ListRuns",
      "awsRegion":"us-west-2",
      "sourceIPAddress":"203.0.113.11",
      "userAgent":"example-user-agent-string",
      "requestParameters": {
        "arn":"arn:aws:devicefarm:us-west-2:123456789012:project:a9129b8c-
df6b-4cdd-8009-40a25EXAMPLE"},
        "responseElements": {
          "runs": [
            {
              "created": "Jul 8, 2015 11:26:12 PM",
              "name": "example.apk",
              "completedJobs": 2,
              "arn": "arn:aws:devicefarm:us-west-2:123456789012:run:a9129b8c-
df6b-4cdd-8009-40a256aEXAMPLE/1452d105-e354-4e53-99d8-6c993EXAMPLE",
              "counters": {
                "stopped": 0,
                "warned": 0,
                "failed": 0,
                "passed": 4,
                "skipped": 0,
                "total": 4,
                "errored": 0
              },
```

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

Integração do AWS Device Farm em um estágio de CodePipeline teste

Você pode usar o AWS CodePipeline para incorporar testes de aplicações móveis configuradas no Device Farm em um pipeline de lançamento automatizado gerenciado pela AWS. Você pode configurar o pipeline para executar testes sob demanda, em uma programação, ou como parte de um fluxo de integração contínua.

O diagrama a seguir mostra o fluxo de integração contínua em que um aplicativo Android é criado e testado cada vez que um envio é confirmado para o repositório. Para criar essa configuração de pipeline, consulte o Tutorial: Crie e teste um aplicativo Android quando enviado para GitHub.



1. Configurar	2. Adicionar definições	3. Push	4. Criar e testar	5. Relatório
Configurar recursos do pipeline	Adicionar definições de criação e teste ao seu pacote	Enviar um pacote para o seu repositório	Compilação e teste de aplicativo de artefato de saída de compilação iniciada automaticamente	Visualizar resultados do teste

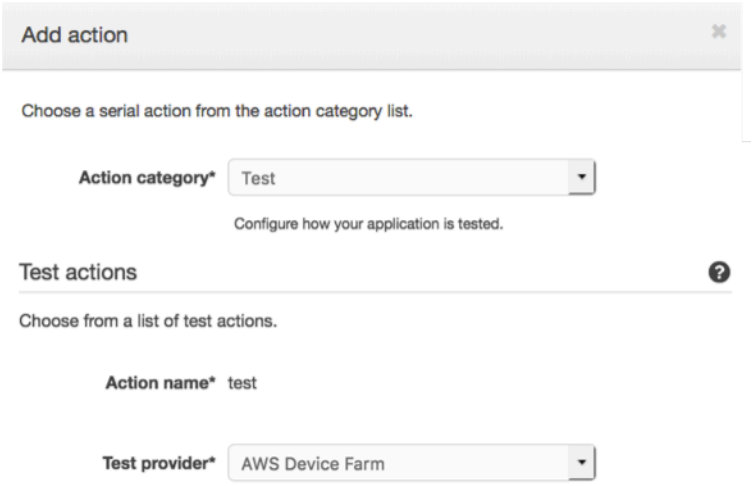
Para saber como configurar um pipeline que testa continuamente um aplicativo compilado (como um arquivo iOS `.ipa` ou arquivo Android `.apk`) como sua origem, consulte Tutorial: Testar um aplicativo iOS toda vez que você carregar um arquivo .ipa em um bucket do Amazon S3.

Configure CodePipeline para usar seus testes do Device Farm

Nessas etapas, presumimos que você tenha configurado um projeto do Device Farm e criado um pipeline. O pipeline deve ser configurado com um estágio de teste que recebe um artefato de entrada que contém a definição do teste e os arquivos do pacote do aplicativo compilado. O artefato de entrada do estágio de teste pode ser o artefato de saída de um estágio de origem ou de compilação configurado no pipeline.

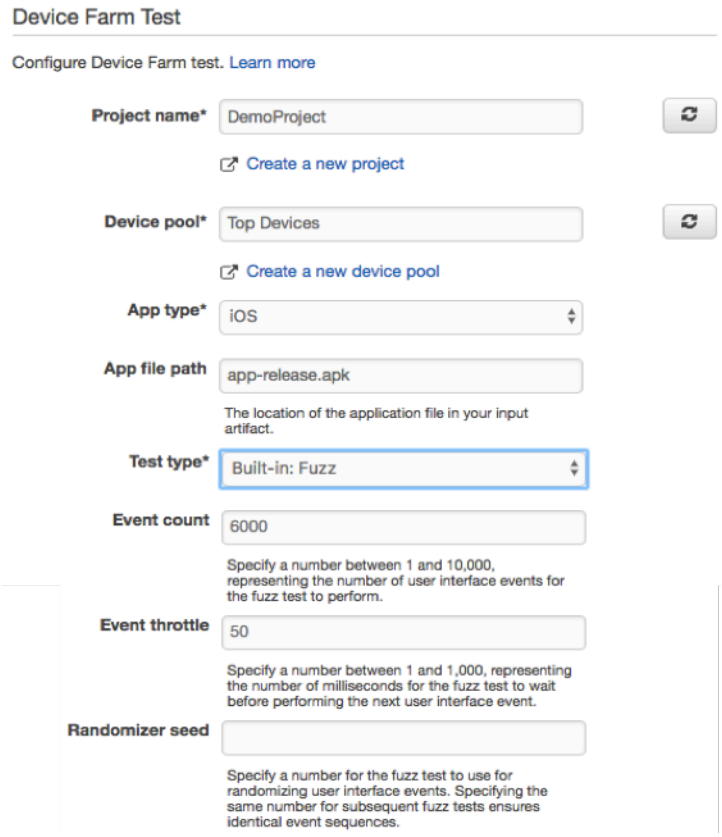
Para configurar um teste do Device Farm, execute como uma ação CodePipeline de teste

1. Faça login no Console de gerenciamento da AWS e abra o CodePipeline console em https://console.aws.amazon.com/codepipeline/.
2. Selecione o pipeline para a versão de seu aplicativo.
3. No painel do estágio de teste, selecione o ícone de lápis e, em seguida, selecione Ação.
4. No painel Adicionar ação, em Categoria da ação, escolha Teste.
5. Em Nome da ação, insira um nome.
6. Em Provedor do teste, selecione AWS Device Farm.



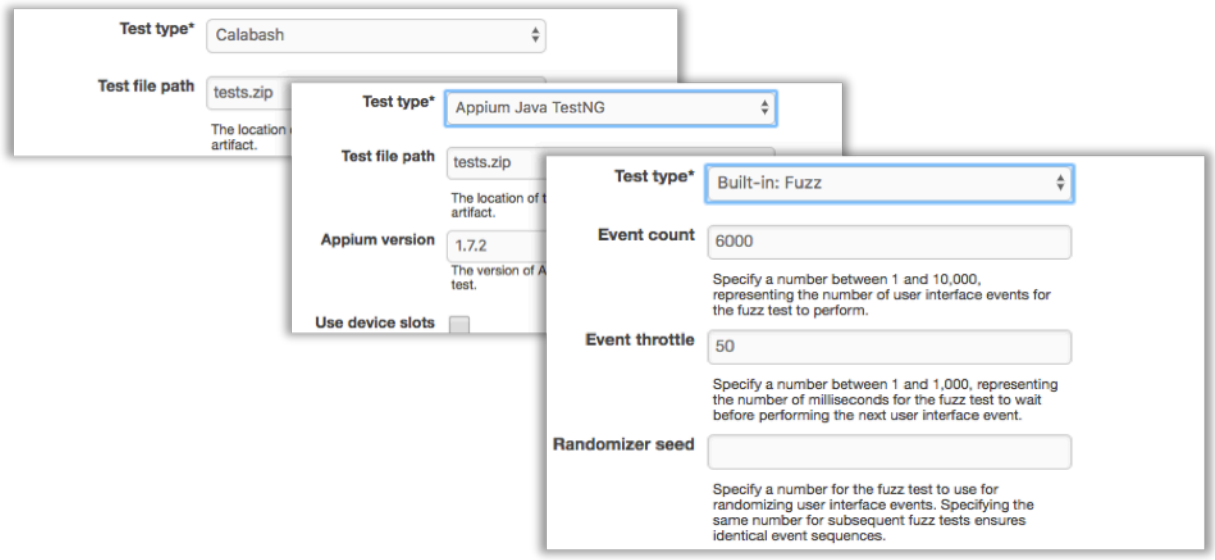
7. Em Nome do projeto, selecione o projeto existente do Device Farm ou selecione Criar um novo projeto.

8. Em Grupo de dispositivos, selecione seu grupo de dispositivos existente ou selecione Criar um novo grupo de dispositivos. Se você criar um grupo de dispositivos, será necessário selecionar um conjunto de dispositivos de teste.
9. Em Tipo de aplicação, selecione a plataforma para sua aplicação.

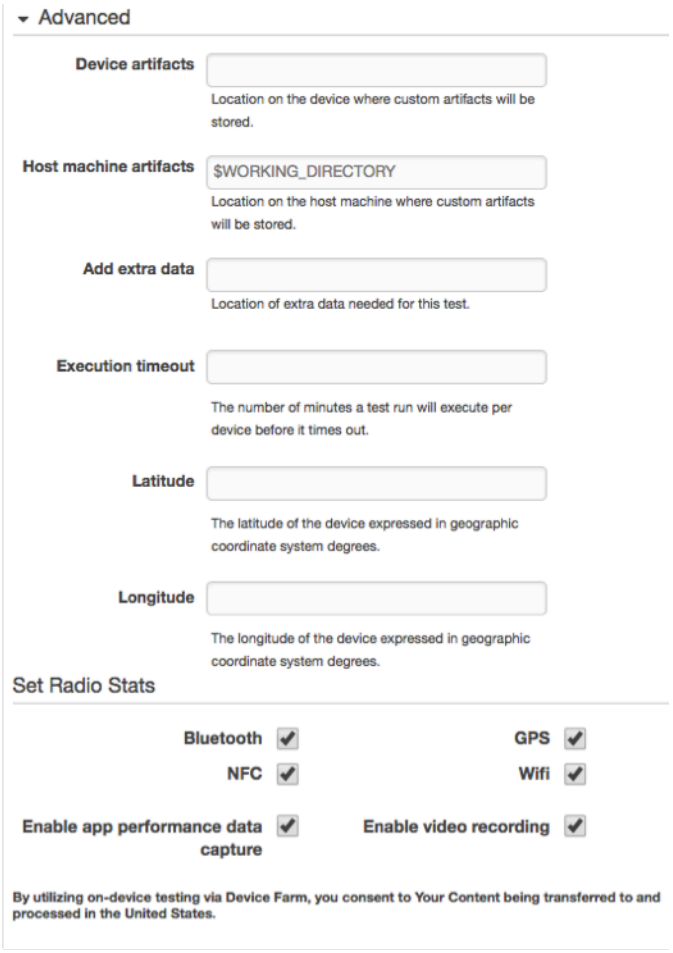


10. Em Caminho para o arquivo da aplicação, insira o caminho do pacote da aplicação compilada. O caminho é relativo à raiz do artefato de entrada para o teste.
11. Em Tipo de teste, siga um destes procedimentos:

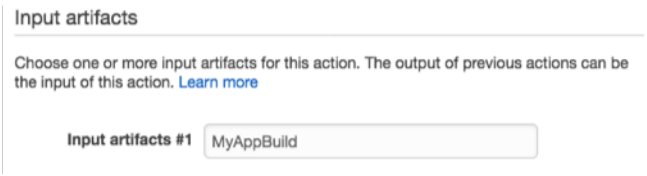
    - Se estiver usando um dos testes internos do Device Farm, escolha o tipo de teste configurado no projeto do Device Farm.
    - Se não estiver usando um dos testes incorporados do Device Farm, no Caminho do arquivo de teste, insira o caminho do arquivo de definição do teste. O caminho é relativo à raiz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



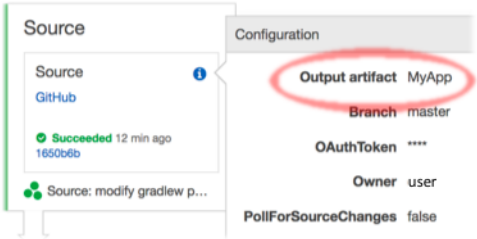
12. Nos campos restantes, forneça a configuração que seja apropriada para seu teste e tipo de aplicativo.
13. (Opcional) Em Avançado, forneça uma configuração detalhada para a execução do teste.



14. Em Artefatos de entrada, selecione o artefato de entrada que corresponde ao artefato de saída do estágio que vem antes do estágio de teste no pipeline.



No CodePipeline console, você pode encontrar o nome do artefato de saída para cada estágio passando o mouse sobre o ícone de informações no diagrama do pipeline. Se seu pipeline testar seu aplicativo diretamente do estágio de origem, escolha MyApp. Se o seu funil incluir um estágio de construção, escolha MyAppBuild.



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

Referência da 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 da AWS Command Line Interface e a Referência de comandos da AWS CLI.

Referência do Windows PowerShell para o AWS Device Farm

Para usar o Windows PowerShell para executar comandos do Device Farm, consulte Device Farm Cmdlet Reference na Referência de Cmdlet do AWS Tools for Windows PowerShell. Para obter mais informações, consulte Setting up the AWS Tools for Windows PowerShell no Guia do usuário do Ferramentas da AWS para 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 AWS CLI forneça meios para fazer isso.

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

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

Exemplo: usar a AWS CLI ou o SDK para carregar um aplicativo ou testar no Device Farm

Os exemplos a seguir mostram como criar um upload no Device Farm usando a AWS CLI ou o AWS SDK em vários idiomas. Os uploads são os principais componentes para programar execuções de teste no Device Farm e incluem o seguinte:

- Seu aplicativo
- Seu teste
- Seu arquivo de especificação de teste

Os uploads são criados usando a `CreateUpload`API. Essa API retorna uma URL pré-assinada do S3 para a qual você pode enviar seu upload usando uma solicitação HTTP PUT. O URL expira após 24 horas.

AWS CLI

> Observação: este exemplo usa a ferramenta de linha de comando *`curl`* para enviar o aplicativo para o Device Farm.
>
> Primeiro, crie um projeto, caso ainda não tenha feito isso.
>
> ```
> $ aws devicefarm create-project --name MyProjectName
> ```
>
> Isso mostrará uma saída como a seguinte:

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

Em seguida, faça o seguinte para criar seu upload e enviá-lo para o Device Farm. Neste exemplo, criaremos um upload de aplicativo Android usando um arquivo APK local. Para obter mais informações sobre o tipo de upload, incluindo detalhes sobre os tipos de upload de aplicativos iOS, consulte nossa documentação de API para criar um `Upload`.

```
$ export APP_PATH="/local/path/to/my_sample_app.apk"
$ export APP_TYPE="ANDROID_APP"
```

Primeiro, criamos o upload no Device Farm:

```
$ aws devicefarm create-upload \
  --project-arn "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE" \
  --name "$(basename "$APP_PATH")" \
  --type "$APP_TYPE"
```

Isso mostrará uma saída como a seguinte:

```
{
    "upload": {
        "arn": "arn:aws:devicefarm:us-
west-2:385076942068:upload:490a6350-0ba3-43e5-83f5-d2896b069a34/a120e848-c57b-4e8d-
a720-d750a0c4d936",
        "name": "my_sample_app.apk",
        "created": 1760747318.266,
        "type": "ANDROID_APP",
        "status": "INITIALIZED",
        "url": "https://prod-us-west-2-uploads.s3.dualstack.us-west-2.amazonaws.com/
arn%3Aaws%3Adevicefarm%3Aus-west-2...",
        "category": "PRIVATE"
    }
}
```

Em seguida, faça uma chamada PUT usando curl para enviar o aplicativo para o bucket S3 do Device Farm:

```
$ curl -T "$APP_PATH" "https://prod-us-west-2-uploads.s3.dualstack.us-
west-2.amazonaws.com/arn%3Aaws%3Adevicefarm%3Aus-west-2..."
```

Por fim, aguarde até que o aplicativo esteja no status “bem-sucedido”:

```
$ aws devicefarm get-upload --arn "arn:aws:devicefarm:us-
west-2:385076942068:upload:490a6350-0ba3-43e5-83f5-d2896b069a34/a120e848-c57b-4e8d-
a720-d750a0c4d936"
```

Isso mostrará uma saída como a seguinte:

```
{
    "upload": {
        "arn": "arn:aws:devicefarm:us-
west-2:385076942068:upload:490a6350-0ba3-43e5-83f5-d2896b069a34/a120e848-c57b-4e8d-
a720-d750a0c4d936",
        "name": "my_sample_app.apk",
        "created": 1760747318.266,
        "type": "ANDROID_APP",
        "status": "SUCCEEDED",
        "url": "https://prod-us-west-2-uploads.s3.dualstack.us-west-2.amazonaws.com/
arn%3Aaws%3Adevicefarm%3Aus-west-2...",
        "metadata": "{\"activity_name\":
\"com.amazonaws.devicefarm.android.referenceapp.Activities.MainActivity\",
\"package_name\":\"com.amazonaws.devicefarm.android.referenceapp\",...}",
        "category": "PRIVATE"
    }
}
```

Python

Observação: este exemplo usa o *`requests`* pacote de terceiros para enviar o aplicativo para o Device Farm, bem como o AWS SDK para Python*`boto3`*.

Primeiro, crie um projeto, caso ainda não tenha feito isso.

```
import boto3

client = boto3.client("devicefarm", region_name="us-west-2")
```

```
resp = client.create_project(name="MyProjectName")

print(resp)
# Response will be something like:
# {
#     "project": {
#         "name": "MyProjectName",
#         "arn": "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE",
#         "created": 1535675814.414
#     }
# }
```

Em seguida, faça o seguinte para criar seu upload e enviá-lo para o Device Farm. Neste exemplo, criaremos um upload de aplicativo Android usando um arquivo APK local. Para obter mais informações sobre o tipo de upload, incluindo detalhes sobre os tipos de upload de aplicativos iOS, consulte nossa documentação de API para criar um `Upload`.

```
import os
import time
import datetime
import requests
from pathlib import Path
import boto3


def upload_device_farm_file():
    project_arn = "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE"
    app_path = Path("/local/path/to/my_sample_app.apk")
    file_type = "ANDROID_APP"

    if not app_path.is_file():
        raise RuntimeError(f"{app_path} is not a valid app file path")

    client = boto3.client("devicefarm", region_name="us-west-2")

    # 1) Create the upload in Device Farm
    create = client.create_upload(
        projectArn=project_arn,
        name=app_path.name,
        type=file_type,
        contentType="application/octet-stream",
```

```
    )
    upload = create["upload"]
    upload_arn = upload["arn"]
    upload_url = upload["url"]
    # This will show output such as the following:
    # { "upload": { "arn": "...", "name": "my_sample_app.apk", "type":
 "ANDROID_APP", "status": "INITIALIZED", "url": "https://..." } }

    # 2) Do an HTTP PUT command to push the file to the pre-signed S3 URL
    with app_path.open("rb") as fh:
        print(f"Uploading {app_path.name} to Device Farm...")
        put_resp = requests.put(upload_url, data=fh, headers={"Content-Type":
 "application/octet-stream"})
        put_resp.raise_for_status()

    # 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
    timeout_seconds = 30
    start = time.time()
    while True:
        get_resp = client.get_upload(arn=upload_arn)
        status = get_resp["upload"]["status"]
        msg = get_resp["upload"].get("message") or
 get_resp["upload"].get("metadata") or ""
        elapsed = datetime.timedelta(seconds=int(time.time() - start))
        print(f"[{elapsed}] status={status}{' - ' + msg if msg else ''}")

        if status == "SUCCEEDED":
            print(f"Upload complete: {upload_arn}")
            return upload_arn
        if status == "FAILED":
            raise RuntimeError(f"Device Farm failed to process upload: {msg}")

        if (time.time() - start) > timeout_seconds:
            raise RuntimeError(f"Timed out after {timeout_seconds}s waiting for
 upload to process (last status={status}).")

        time.sleep(1)

upload_device_farm_file()
```

Java

Observação: este exemplo usa o AWS SDK for Java v2 *`HttpClient`* e para enviar o aplicativo para o Device Farm e é compatível com as versões 11 e posteriores do JDK.

Primeiro, crie um projeto, caso ainda não tenha feito isso.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import software.amazon.awssdk.services.devicefarm.model.CreateProjectRequest;
import software.amazon.awssdk.services.devicefarm.model.CreateProjectResponse;

try (DeviceFarmClient client = DeviceFarmClient.builder()
        .region(Region.US_WEST_2)
        .build()) {
    CreateProjectResponse resp = client.createProject(
        CreateProjectRequest.builder().name("MyProjectName").build());
    System.out.println(resp.project());
    // Response will be something like:
    // Project{name=MyProjectName, arn=arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-..., created=...}
}
```

Em seguida, faça o seguinte para criar seu upload e enviá-lo para o Device Farm. Neste exemplo, criaremos um upload de aplicativo Android usando um arquivo APK local. Para obter mais informações sobre o tipo de upload, incluindo detalhes sobre os tipos de upload de aplicativos iOS, consulte nossa documentação de API para criar um `Upload`.

```
import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.time.Duration;
import java.time.Instant;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import software.amazon.awssdk.services.devicefarm.model.CreateUploadRequest;
import software.amazon.awssdk.services.devicefarm.model.CreateUploadResponse;
import software.amazon.awssdk.services.devicefarm.model.GetUploadRequest;
import software.amazon.awssdk.services.devicefarm.model.GetUploadResponse;
import software.amazon.awssdk.services.devicefarm.model.Upload;
import software.amazon.awssdk.services.devicefarm.model.UploadType;
```

```
public class DeviceFarmUploader {

    public static String upload(String projectArn, Path appPath) throws Exception {
        if (projectArn == null || projectArn.isEmpty()) {
            throw new IllegalArgumentException("Missing projectArn");
        }
        if (!Files.isRegularFile(appPath)) {
            throw new IllegalArgumentException("Invalid app path: " + appPath);
        }

        String fileName = appPath.getFileName().toString().trim();
        UploadType type = UploadType.ANDROID_APP;

        // Build a reusable HttpClient
        HttpClient http = HttpClient.newBuilder()
                .version(HttpClient.Version.HTTP_1_1)
                .connectTimeout(Duration.ofSeconds(10))
                .build();

        try (DeviceFarmClient client = DeviceFarmClient.builder()
                .region(Region.US_WEST_2)
                .build()) {

            // 1) Create the upload in Device Farm
            CreateUploadResponse create =
 client.createUpload(CreateUploadRequest.builder()
                    .projectArn(projectArn)
                    .name(fileName)
                    .type(type)
                    .contentType("application/octet-stream")
                    .build());

            Upload upload = create.upload();
            String uploadArn = upload.arn();
            String url = upload.url();
            // This will show output such as the following:
            // { "upload": { "arn": "...", "name": "my_sample_app.apk", "type":
 "ANDROID_APP", "status": "INITIALIZED", "url": "https://..." } }

            // 2) PUT file to pre-signed URL using HttpClient
            HttpRequest put = HttpRequest.newBuilder(URI.create(url))
                    .timeout(Duration.ofMinutes(15))
                    .header("Content-Type", "application/octet-stream")
```

```
                    .PUT(HttpRequest.BodyPublishers.ofFile(appPath))
                    .build();

            HttpResponse<Void> resp = http.send(put,
 HttpResponse.BodyHandlers.discarding());
            int code = resp.statusCode();
            if (code / 100 != 2) {
                throw new IOException("Failed PUT to S3 pre-signed URL, HTTP " +
 code);
            }

            // 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
            Instant deadline = Instant.now().plusSeconds(30); // 30-second timeout
            while (true) {
                GetUploadResponse got = client.getUpload(GetUploadRequest.builder()
                        .arn(uploadArn)
                        .build());

                String status = got.upload().statusAsString();
                String msg = got.upload().metadata();
                System.out.println("status=" + status + (msg != null ? " - " + msg :
 ""));

                if ("SUCCEEDED".equals(status)) return uploadArn;
                if ("FAILED".equals(status)) throw new RuntimeException("Upload
 failed: " + msg);
                if (Instant.now().isAfter(deadline)) {
                    throw new RuntimeException("Timeout waiting for processing, last
 status=" + status);
                }
                Thread.sleep(2000);
            }
        }
    }

    public static void main(String[] args) throws Exception {
        String projectArn = "arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-c861-4c0a-b1d5-12345EXAMPLE";
        Path appPath = Paths.get("/local/path/to/my_sample_app.apk");
        String result = upload(projectArn, appPath);
        System.out.println("Upload ARN: " + result);
    }
}
```

JavaScript

Observação: este exemplo usa o AWS SDK for JavaScript (v3) e o Node 18+ *fetch* para enviar o aplicativo para o Device Farm.

Primeiro, crie um projeto, caso ainda não tenha feito isso.

```
import { DeviceFarmClient, CreateProjectCommand } from "@aws-sdk/client-device-
farm";

const df = new DeviceFarmClient({ region: "us-west-2" });
const resp = await df.send(new CreateProjectCommand({ name: "MyProjectName" }));
console.log(resp);
// Response will be something like:
// { project: { name: 'MyProjectName', arn: 'arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-...', created: 1535675814.414 } }
```

Em seguida, faça o seguinte para criar seu upload e enviá-lo para o Device Farm. Neste exemplo, criaremos um upload de aplicativo Android usando um arquivo APK local. Para obter mais informações sobre o tipo de upload, incluindo detalhes sobre os tipos de upload de aplicativos iOS, consulte nossa documentação de API para criar um `Upload`.

```
import { DeviceFarmClient, CreateUploadCommand, GetUploadCommand } from "@aws-sdk/
client-device-farm";
import { createReadStream } from "fs";
import { basename } from "path";

const projectArn = "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE";
const appPath = "/local/path/to/my_sample_app.apk";
const name = basename(appPath).trim();
const type = "ANDROID_APP";

const client = new DeviceFarmClient({ region: "us-west-2" });

// 1) Create the upload in Device Farm
const create = await client.send(new CreateUploadCommand({
  projectArn,
  name,
  type,
  contentType: "application/octet-stream",
}));
```

```
const uploadArn = create.upload.arn;
const url = create.upload.url;
// This will show output such as the following:
// { upload: { arn: '...', name: 'my_sample_app.apk', type: 'ANDROID_APP', status:
 'INITIALIZED', url: 'https://...' } }

// 2) PUT to pre-signed URL
const putResp = await fetch(url, {
  method: "PUT",
  headers: { "Content-Type": "application/octet-stream" },
  body: createReadStream(appPath),
});
if (!putResp.ok) {
  throw new Error(`Failed PUT to pre-signed URL: ${putResp.status} ${await
 putResp.text().catch(()=>"")}`);
}

// 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
const deadline = Date.now() + (30 * 1000); // 30-second timeout
while (true) {
  const response = await client.send(new GetUploadCommand({ arn: uploadArn }));
  const { status, message, metadata } = response.upload;
  console.log(`status=${status}${message ? " - " + message : metadata ? " - " +
 metadata : ""}`);
  if (status === "SUCCEEDED") {
    console.log("Upload complete:", uploadArn);
    break;
  }
  if (status === "FAILED") {
    throw new Error(`Upload failed: ${message || metadata || "unknown"}`);
  }
  if (Date.now() > deadline) throw new Error(`Timeout waiting for processing (last
 status=${status})`);
  await new Promise(r => setTimeout(r, 2000));
}
```

C#

Observação: este exemplo usa o AWS SDK para.NET *`HttpClient`* e para enviar o aplicativo para o Device Farm.

Primeiro, crie um projeto, caso ainda não tenha feito isso.

```
using System;
```

```
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

using var client = new AmazonDeviceFarmClient(RegionEndpoint.USWest2);
var resp = await client.CreateProjectAsync(new CreateProjectRequest { Name =
 "MyProjectName" });
Console.WriteLine(resp.Project);
// Response will be something like:
// { Name = MyProjectName, Arn = arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-..., Created = ... }
```

Em seguida, faça o seguinte para criar seu upload e enviá-lo para o Device Farm. Neste exemplo, criaremos um upload de aplicativo Android usando um arquivo APK local. Para obter mais informações sobre o tipo de upload, incluindo detalhes sobre os tipos de upload de aplicativos iOS, consulte nossa documentação de API para criar um `Upload`.

```
using System;
using System.IO;
using System.Net.Http;
using System.Threading.Tasks;
using System.Net.Http.Headers;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

class DeviceFarmUploader
{
    public static async Task<string> UploadAsync(string projectArn, string appPath)
    {
        if (string.IsNullOrWhiteSpace(projectArn)) throw new
 ArgumentException("Missing projectArn");
        if (!File.Exists(appPath)) throw new ArgumentException($"Invalid app path:
 {appPath}");
        var type = UploadType.ANDROID_APP;

        using var client = new AmazonDeviceFarmClient(RegionEndpoint.USWest2);
        // 1) Create the upload in Device Farm
        var create = await client.CreateUploadAsync(new CreateUploadRequest
        {
            ProjectArn = projectArn,
            Name = Path.GetFileName(appPath),
            Type = type,
```

```
            ContentType = "application/octet-stream"
        });

        var uploadArn = create.Upload.Arn;
        var url = create.Upload.Url;
        // This will show output such as the following:
        // { Upload: { Arn = ..., Name = my_sample_app.apk, Type = ANDROID_APP,
Status = INITIALIZED, Url = https://... } }

        // 2) PUT file to pre-signed URL
        using (var http = new HttpClient())
        using (var fs = File.OpenRead(appPath))
        using (var content = new StreamContent(fs))
        {
            content.Headers.Add("Content-Type", "application/octet-stream");
            var resp = await http.PutAsync(url, content);
            if (!resp.IsSuccessStatusCode)
                throw new Exception($"Failed PUT to pre-signed URL:
{(int)resp.StatusCode} {await resp.Content.ReadAsStringAsync()}");
        }

        // 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
        var deadline = DateTime.UtcNow.AddSeconds(30); // 30-second timeout
        while (true)
        {
            var got = await client.GetUploadAsync(new GetUploadRequest { Arn =
uploadArn });
            var status = got.Upload.Status.Value;
            var msg = got.Upload.Message ?? got.Upload.Metadata;
            Console.WriteLine($"status={status}{(string.IsNullOrEmpty(msg) ? "" : "
- " + msg)}");

            if (status == UploadStatus.SUCCEEDED.Value) return uploadArn;
            if (status == UploadStatus.FAILED.Value) throw new Exception($"Upload
failed: {msg}");
            if (DateTime.UtcNow > deadline) throw new TimeoutException($"Timeout
waiting for processing (last status={status})");
            await Task.Delay(2000);
        }
    }

    static async Task Main()
    {
```

```
        var projectArn = "arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-c861-4c0a-b1d5-12345EXAMPLE";
        var appPath = "/local/path/to/my_sample_app.apk";
        var result = await UploadAsync(projectArn!, appPath!);
        Console.WriteLine("Upload ARN: " + result);
    }
}
```

Ruby

Observação: este exemplo usa o AWS SDK for *Net::HTTP* Ruby e para enviar o aplicativo para o Device Farm.

Primeiro, crie um projeto, caso ainda não tenha feito isso.

```
require "aws-sdk-devicefarm"

client = Aws::DeviceFarm::Client.new(region: "us-west-2")
resp = client.create_project(name: "MyProjectName")
puts resp.project.inspect
# Response will be something like:
# #<struct Aws::DeviceFarm::Types::Project name="MyProjectName",
 arn="arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-...",
 created=1535675814.414>
```

Em seguida, faça o seguinte para criar seu upload e enviá-lo para o Device Farm. Neste exemplo, criaremos um upload de aplicativo Android usando um arquivo APK local. Para obter mais informações sobre o tipo de upload, incluindo detalhes sobre os tipos de upload de aplicativos iOS, consulte nossa documentação de API para criar um `Upload`.

```
require "aws-sdk-devicefarm"
require "net/http"
require "uri"

project_arn = "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-c861-4c0a-
b1d5-12345EXAMPLE"
app_path    = "/local/path/to/my_sample_app.apk"
raise "Invalid APP_PATH: #{app_path}" unless File.file?(app_path)
type = "ANDROID_APP"

client = Aws::DeviceFarm::Client.new(region: "us-west-2")

# 1) Create the upload in Device Farm
```

```
create = client.create_upload(
  project_arn: project_arn,
  name: File.basename(app_path),
  type: type,
  content_type: "application/octet-stream"
)

upload_arn = create.upload.arn
url = create.upload.url
# This will show output such as the following:
# #<Upload arn="...", name="my_sample_app.apk", type="ANDROID_APP",
 status="INITIALIZED", url="https://...">

# 2) PUT the file to the pre-signed URL
uri = URI.parse(url)
Net::HTTP.start(uri.host, uri.port, use_ssl: (uri.scheme == "https")) do |http|
  req = Net::HTTP::Put.new(uri)
  req["Content-Type"] = "application/octet-stream"
  req.body_stream = File.open(app_path, "rb")
  req.content_length = File.size(app_path)
  resp = http.request(req)
  raise "Failed PUT: #{resp.code} #{resp.body}" unless resp.code.to_i / 100 == 2
end

# 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
deadline = Time.now + 30 # 30-second timeout
loop do
  got = client.get_upload(arn: upload_arn)
  status = got.upload.status
  msg = got.upload.message || got.upload.metadata
  puts "status=#{status}#{msg ? " - #{msg}" : ""}"

  case status
  when "SUCCEEDED" then puts "Upload complete: #{upload_arn}"; break
  when "FAILED"    then raise "Upload failed: #{msg}"
  end
  raise "Timeout waiting for processing (last status=#{status})" if Time.now >
 deadline
  sleep 2
end
```

Exemplo: usar o AWS SDK para iniciar uma execução do Device Farm e coletar artefatos

O exemplo a seguir beginning-to-end mostra como você pode usar o AWS SDK para trabalhar com o Device Farm. Esse exemplo faz o seguinte:

- Carrega um teste e pacotes de aplicações para o Device Farm
- Inicia uma execução de teste e aguarda sua conclusão (ou falha)
- Faz download de todos os artefatos produzidos pelos conjuntos de testes

Esse exemplo depende do pacote `requests` de terceiros para interagir com HTTP.

```
import boto3
import os
import requests
import string
import random
import time
import datetime
import time
import json

# The following script runs a test through Device Farm
#
# Things you have to change:
config = {
    # This is our app under test.
    "appFilePath":"app-debug.apk",
    "projectArn": "arn:aws:devicefarm:us-
west-2:111122223333:project:1b99bcff-1111-2222-ab2f-8c3c733c55ed",
    # Since we care about the most popular devices, we'll use a curated pool.
    "testSpecArn":"arn:aws:devicefarm:us-west-2::upload:101e31e8-12ac-11e9-ab14-
d663bd873e83",
    "poolArn":"arn:aws:devicefarm:us-west-2::devicepool:082d10e5-d7d7-48a5-ba5c-
b33d66efa1f5",
    "namePrefix":"MyAppTest",
    # This is our test package. This tutorial won't go into how to make these.
    "testPackage":"tests.zip"
}
```

```
client = boto3.client('devicefarm')

unique =
 config['namePrefix']+"-"+(datetime.date.today().isoformat())+(''.join(random.sample(string.asc

print(f"The unique identifier for this run is going to be {unique} -- all uploads will
 be prefixed with this.")

def upload_df_file(filename, type_, mime='application/octet-stream'):
    response = client.create_upload(projectArn=config['projectArn'],
        name = (unique)+"_"+os.path.basename(filename),
        type=type_,
        contentType=mime
        )
    # Get the upload ARN, which we'll return later.
    upload_arn = response['upload']['arn']
    # We're going to extract the URL of the upload and use Requests to upload it
    upload_url = response['upload']['url']
    with open(filename, 'rb') as file_stream:
        print(f"Uploading {filename} to Device Farm as {response['upload']['name']}...
 ",end='')
        put_req = requests.put(upload_url, data=file_stream, headers={"content-
type":mime})
        print(' done')
        if not put_req.ok:
            raise Exception("Couldn't upload, requests said we're not ok. Requests
 says: "+put_req.reason)
    started = datetime.datetime.now()
    while True:
        print(f"Upload of {filename} in state {response['upload']['status']} after
 "+str(datetime.datetime.now() - started))
        if response['upload']['status'] == 'FAILED':
            raise Exception("The upload failed processing. DeviceFarm says reason
 is: \n"+(response['upload']['message'] if 'message' in response['upload'] else
 response['upload']['metadata']))
        if response['upload']['status'] == 'SUCCEEDED':
            break
        time.sleep(5)
        response = client.get_upload(arn=upload_arn)
    print("")
    return upload_arn

our_upload_arn = upload_df_file(config['appFilePath'], "ANDROID_APP")
```

```
our_test_package_arn = upload_df_file(config['testPackage'],
 'APPIUM_PYTHON_TEST_PACKAGE')
print(our_upload_arn, our_test_package_arn)
# Now that we have those out of the way, we can start the test run...
response = client.schedule_run(
    projectArn = config["projectArn"],
    appArn = our_upload_arn,
    devicePoolArn = config["poolArn"],
    name=unique,
    test = {
        "type":"APPIUM_PYTHON",
        "testSpecArn": config["testSpecArn"],
        "testPackageArn": our_test_package_arn
        }
    )
run_arn = response['run']['arn']
start_time = datetime.datetime.now()
print(f"Run {unique} is scheduled as arn {run_arn} ")

try:

    while True:
        response = client.get_run(arn=run_arn)
        state = response['run']['status']
        if state == 'COMPLETED' or state == 'ERRORED':
            break
        else:
            print(f" Run {unique} in state {state}, total time
 "+str(datetime.datetime.now()-start_time))
            time.sleep(10)
except:
    # If something goes wrong in this process, we stop the run and exit.

    client.stop_run(arn=run_arn)
    exit(1)
print(f"Tests finished in state {state} after "+str(datetime.datetime.now() -
 start_time))
# now, we pull all the logs.
jobs_response = client.list_jobs(arn=run_arn)
# Save the output somewhere. We're using the unique value, but you could use something
 else
save_path = os.path.join(os.getcwd(), unique)
os.mkdir(save_path)
# Save the last run information
```

```
for job in jobs_response['jobs'] :
    # Make a directory for our information
    job_name = job['name']
    os.makedirs(os.path.join(save_path, job_name), exist_ok=True)
    # Get each suite within the job
    suites = client.list_suites(arn=job['arn'])['suites']
    for suite in suites:
        for test in client.list_tests(arn=suite['arn'])['tests']:
            # Get the artifacts
            for artifact_type in ['FILE','SCREENSHOT','LOG']:
                artifacts = client.list_artifacts(
                    type=artifact_type,
                    arn = test['arn']
                )['artifacts']
                for artifact in artifacts:
                    # We replace : because it has a special meaning in Windows & macos
                    path_to = os.path.join(save_path, job_name, suite['name'],
 test['name'].replace(':','_') )
                    os.makedirs(path_to, exist_ok=True)
                    filename =
 artifact['type']+"_"+artifact['name']+"."+artifact['extension']
                    artifact_save_path = os.path.join(path_to, filename)
                    print("Downloading "+artifact_save_path)
                    with open(artifact_save_path, 'wb') as fn,
 requests.get(artifact['url'],allow_redirects=True) as request:
                        fn.write(request.content)
                    #/for artifact in artifacts
                #/for artifact type in []
            #/ for test in ()[]
        #/ for suite in suites
    #/ for job in _[]
# done
print("Finished")
```

Solução de problemas de Device Farm

Nesta seção, você encontrará mensagens de erro e procedimentos para ajudá-lo a corrigir problemas comuns com o Device Farm.

> **Note**
>
> Para solucionar problemas de testes do Appium que falham inesperadamente no Device Farm, consulte nosso guia para testes do Appium do lado do cliente

Tópicos

Solução de problemas de testes de aplicações para Android no AWS Device Farm

O tópico a seguir lista mensagens de erro que ocorrem durante o upload de testes de aplicativos Android e recomenda soluções para resolver cada erro.

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

ANDROID_APP_UNZIP_FAILED

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

> **Warning**
>
> Não conseguimos abrir seu aplicativo. Verifique se o arquivo é válido e tente novamente.

Verifique se você consegue descompactar o pacote de aplicativos sem erros. No exemplo a seguir, o nome do pacote é app-debug.apk.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip app-debug.apk
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Um pacote de aplicativos Android válido deve gerar um resultado semelhante ao seguinte:

   ```
   .
   |-- AndroidManifest.xml
   |-- classes.dex
   |-- resources.arsc
   |-- assets (directory)
   |-- res (directory)
   `-- META-INF (directory)
   ```

   Para obter mais informações, consulte Testes 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 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 pela saída de um comando `aapt debug badging` *<path to your package>*.

Verifique se você consegue executar esse comando com êxito em seu aplicativo Android e encontrar o valor do nome do pacote. No exemplo a seguir, o nome do pacote é app-debug.apk.

- Copie o pacote de aplicativos para seu diretório de trabalho e execute o seguinte comando:

  ```
  $ aapt debug badging app-debug.apk | grep "package: name="
  ```

  Um pacote de aplicativos Android válido deve gerar um resultado semelhante ao seguinte:

  ```
  package: name='com.amazon.aws.adf.android.referenceapp' versionCode='1'
   versionName='1.0' platformBuildVersionName='5.1.1-1819727'
  ```

  Para obter mais informações, consulte Testes 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 pela saída de um comando `aapt debug badging` *<path to your package>*.

Verifique se você consegue executar esse comando com êxito em seu aplicativo Android e encontrar o valor do nome do pacote. No exemplo a seguir, o nome do pacote é app-debug.apk.

- Copie o pacote de aplicativos para seu diretório de trabalho e execute o seguinte comando:

  ```
  $ aapt debug badging app-debug.apk | grep "sdkVersion"
  ```

  Um pacote de aplicativos Android válido deve gerar um resultado semelhante ao seguinte:

  ```
  sdkVersion:'9'
  ```

  Para obter mais informações, consulte Testes 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 arquivo.xml válido em seu aplicativo. Para verificar se o pacote de testes é válido, execute o comando `aapt dump xmltree` *<path to your test package>* `AndroidManifest.xml` e tente novamente se o comando não imprimir nenhum erro.

Durante o processo de validação de upload, o AWS Device Farm analisa as informações da árvore de análise XML para um arquivo XML contido no pacote usando o comando `aapt dump xmltree` *<path to your package>* `AndroidManifest.xml`.

Verifique se você consegue executar esse comando com êxito em seu aplicativo Android. No exemplo a seguir, o nome do pacote é app-debug.apk.

- Copie o pacote de aplicativos para seu diretório de trabalho e execute o seguinte comando:

```
$ aapt dump xmltree app-debug.apk. AndroidManifest.xml
```

Um pacote de aplicativos Android válido deve gerar um resultado semelhante ao seguinte:

```
N: android=http://schemas.android.com/apk/res/android
  E: manifest (line=2)
    A: android:versionCode(0x0101021b)=(type 0x10)0x1
    A: android:versionName(0x0101021c)="1.0" (Raw: "1.0")
    A: package="com.amazon.aws.adf.android.referenceapp" (Raw:
 "com.amazon.aws.adf.android.referenceapp")
    A: platformBuildVersionCode=(type 0x10)0x16 (Raw: "22")
    A: platformBuildVersionName="5.1.1-1819727" (Raw: "5.1.1-1819727")
    E: uses-sdk (line=7)
      A: android:minSdkVersion(0x0101020c)=(type 0x10)0x9
      A: android:targetSdkVersion(0x01010270)=(type 0x10)0x16
    E: uses-permission (line=11)
      A: android:name(0x01010003)="android.permission.INTERNET" (Raw:
 "android.permission.INTERNET")
    E: uses-permission (line=12)
      A: android:name(0x01010003)="android.permission.CAMERA" (Raw:
 "android.permission.CAMERA")
```

Para obter mais informações, consulte Testes 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.xml` e tente novamente assim que confirmar que a saída não contém a palavra-chave `android.permission.BIND_DEVICE_ADMIN`.

Durante o processo de validação de upload, o AWS Device Farm analisa as informações de permissão da árvore de análise xml para um arquivo xml contido no pacote usando o comando `aapt dump xmltree <path to your package> AndroidManifest.xml`.

Descobrimos que seu aplicativo não requer permissão de administrador do dispositivo. No exemplo a seguir, o nome do pacote é app-debug.apk.

- Copie o pacote de aplicativos para seu diretório de trabalho e execute o seguinte comando:

```
$ aapt dump xmltree app-debug.apk AndroidManifest.xml
```

Você provavelmente chegará a um resultado como o seguinte:

```
N: android=http://schemas.android.com/apk/res/android
  E: manifest (line=2)
    A: android:versionCode(0x0101021b)=(type 0x10)0x1
    A: android:versionName(0x0101021c)="1.0" (Raw: "1.0")
    A: package="com.amazonaws.devicefarm.android.referenceapp" (Raw:
 "com.amazonaws.devicefarm.android.referenceapp")
    A: platformBuildVersionCode=(type 0x10)0x16 (Raw: "22")
    A: platformBuildVersionName="5.1.1-1819727" (Raw: "5.1.1-1819727")
    E: uses-sdk (line=7)
      A: android:minSdkVersion(0x0101020c)=(type 0x10)0xa
      A: android:targetSdkVersion(0x01010270)=(type 0x10)0x16
    E: uses-permission (line=11)
      A: android:name(0x01010003)="android.permission.INTERNET" (Raw:
 "android.permission.INTERNET")
    E: uses-permission (line=12)
      A: android:name(0x01010003)="android.permission.CAMERA" (Raw:
 "android.permission.CAMERA")
        ……
```

Se o aplicativo Android for válido, a saída não deverá conter o seguinte: `A: android:name(0x01010003)="android.permission.BIND_DEVICE_ADMIN" (Raw: "android.permission.BIND_DEVICE_ADMIN")`.

Para obter mais informações, consulte Testes 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 recurso `FLAG_SECURE` do Android. Esse sinalizador (conforme descrito na documentação oficial do Android) é usado para impedir que determinadas janelas de uma aplicação sejam gravadas por ferramentas de gravação de tela. Como resultado, o recurso de gravação de tela do Device Farm (para testes de automação e acesso remoto) pode mostrar uma tela preta no lugar da janela da sua aplicação se a janela usar esse sinalizador.

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

Além disso, observe que o Device Farm ainda pode interagir com janelas de aplicações que tenham esse sinalizador. Portanto, se a página de login da aplicação aparecer como uma tela preta, você ainda poderá inserir suas credenciais para fazer login na aplicação (e, assim, visualizar páginas não bloqueadas pelo sinalizador `FLAG_SECURE`).

Solução de problemas de 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_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 Execute testes Appium automaticamente no 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 Execute testes Appium automaticamente no 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 Execute testes Appium automaticamente no 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 Execute testes Appium automaticamente no 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 Execute testes Appium automaticamente no 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 Execute testes Appium automaticamente no 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 Execute testes Appium automaticamente no Device Farm.

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

O tópico a seguir lista as mensagens de erro que ocorrem durante o upload dos testes do aplicativo Appium Java JUnit Web e recomenda soluções alternativas para resolver cada erro. Para obter mais informações sobre como usar o Appium com o Device Farm, consulte the section called “Testes automáticos de appium”.

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED

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

> **Warning**
>
> Não conseguimos abrir seu arquivo de teste ZIP. Verifique se o arquivo é válido e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Um JUnit pacote Appium Java válido deve produzir uma saída como a seguinte:

   ```
   .
   ```

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

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_DEPENDENCY_DIR_MISSING

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

> **Warning**
>
> Não foi possível encontrar o diretório dependency-jars em seu pacote de testes. Descompacte o pacote de testes, verifique se o diretório dependency-jars encontra-se no pacote e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Se o JUnit pacote Appium Java for válido, você encontrará o *`dependency-jars`* diretório dentro do diretório de trabalho:

   ```
   .
   |— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
   ```

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

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_IN_DEPENDEN

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

> **Warning**
>
> Não foi possível localizar um arquivo JAR na árvore do diretório dependency-jars. Descompacte o pacote de testes e abra o diretório dependency-jars, verifique se pelo menos um arquivo JAR encontra-se no diretório e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Se o JUnit pacote Appium Java for válido, você encontrará pelo menos um *jar* arquivo dentro do *dependency-jars* diretório:

   ```
   .
   |— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
   |— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
   ```

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

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

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

> **Warning**
>
> Não foi possível encontrar um arquivo * -tests.jar em seu pacote de testes. Descompacte o pacote de testes, verifique se pelo menos um arquivo * -tests.jar encontra-se no pacote e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Se o JUnit pacote Appium Java for válido, você encontrará pelo menos um *jar* arquivo, como *acme-android-appium-1.0-SNAPSHOT-tests.jar* no nosso exemplo. O nome do arquivo pode ser diferente, mas deve terminar com*-tests.jar*.

   ```
   .
   |– acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
   |– acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
   ```

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

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TES

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

> **Warning**
>
> Não foi possível localizar um arquivo de classe no arquivo de testes JAR. Descompacte o pacote de testes e o arquivo de testes JAR, verifique se pelo menos um arquivo de classe encontra-se no arquivo JAR e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Você deve encontrar pelo menos um arquivo jar, como *acme-android-appium-1.0-SNAPSHOT-tests.jar* no nosso exemplo. O nome do arquivo pode ser diferente, mas deve terminar com*-tests.jar*.

   ```
   .
   |– acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
    built from the ./src/main directory)
   |– acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
    everything built from the ./src/test directory)
   ```

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

3. Assim que conseguir extrair os arquivos, deverá encontrar pelo menos uma classe na árvore do diretório de trabalho executando o comando:

```
$ tree .
```

Você deve ver um resultado semelhante a este:

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

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_JUNIT_VERSION_VALUE_UNK

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

> **Warning**
>
> Não foi possível encontrar um valor de JUnit versão. Descompacte seu pacote de teste e abra o diretório dependency-jars, verifique se o arquivo JUnit JAR está dentro do diretório e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip zip-with-dependencies.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
tree .
```

   O resultado deve ser semelhante ao seguinte:

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— junit-4.10.jar
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

   Se o JUnit pacote Appium Java for válido, você encontrará o arquivo de JUnit dependência semelhante ao arquivo jar *junit-4.10.jar* em nosso exemplo. O nome deve consistir na palavra-chave *junit* e em seu número de versão, que neste exemplo é 4.10.

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION

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

> **Warning**
>
> Descobrimos que a JUnit versão era inferior à versão mínima 4.10 que suportamos. Altere a JUnit versão e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip zip-with-dependencies.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Você deve encontrar um arquivo de JUnit dependência como *junit-4.10.jar* em nosso exemplo e seu número de versão, que em nosso exemplo é 4.10:

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— junit-4.10.jar
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

> **Note**
>
> Seus testes podem não ser executados corretamente se a JUnit versão especificada em seu pacote de teste for inferior à versão mínima 4.10 que suportamos.

Para obter mais informações, consulte Execute testes Appium automaticamente no 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 Execute testes Appium automaticamente no 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 Execute testes Appium automaticamente no 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 Execute testes Appium automaticamente no 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 Execute testes Appium automaticamente no 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 Execute testes Appium automaticamente no Device Farm.

Solução de problemas de aplicações Web Appium Java TestNG no AWS Device Farm

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

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED

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

> **Warning**
>
> Não conseguimos abrir seu arquivo de teste ZIP. Verifique se o arquivo é válido e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Um JUnit pacote Appium Java válido deve produzir uma saída como a seguinte:

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

Para obter mais informações, consulte Execute testes Appium automaticamente no Device Farm.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSIN

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

> **Warning**
>
> Não foi possível encontrar o diretório dependency-jars em seu pacote de testes. Descompacte o pacote de testes, verifique se o diretório dependency-jars encontra-se no pacote e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

Se o JUnit pacote Appium Java for válido, você encontrará o *dependency-jars* diretório dentro do diretório de trabalho.

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

Para obter mais informações, consulte Execute testes Appium automaticamente no Device Farm.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPEND

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

> **Warning**
>
> Não foi possível localizar um arquivo JAR na árvore do diretório dependency-jars. Descompacte o pacote de testes e abra o diretório dependency-jars, verifique se pelo menos um arquivo JAR encontra-se no diretório e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o JUnit pacote Appium Java for válido, você encontrará pelo menos um *jar* arquivo dentro do *dependency-jars* diretório.

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

Para obter mais informações, consulte Execute testes Appium automaticamente no Device Farm.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

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

> **Warning**
>
> Não foi possível encontrar um arquivo * -tests.jar em seu pacote de testes. Descompacte o pacote de testes, verifique se pelo menos um arquivo * -tests.jar encontra-se no pacote e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip zip-with-dependencies.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o JUnit pacote Appium Java for válido, você encontrará pelo menos um *jar* arquivo, como *acme-android-appium-1.0-SNAPSHOT-tests.jar* no nosso exemplo. O nome do arquivo pode ser diferente, mas deve terminar com*-tests.jar*.

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

Para obter mais informações, consulte Execute testes Appium automaticamente no Device Farm.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_T

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

> **Warning**
>
> Não foi possível localizar um arquivo de classe no arquivo de testes JAR. Descompacte o pacote de testes e o arquivo de testes JAR, verifique se pelo menos um arquivo de classe encontra-se no arquivo JAR e tente novamente.

No exemplo a seguir, o nome do pacote é zip-with-dependencies.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip zip-with-dependencies.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Você deve encontrar pelo menos um arquivo jar, como *acme-android-appium-1.0-SNAPSHOT-tests.jar* no nosso exemplo. O nome do arquivo pode ser diferente, mas deve terminar com*-tests.jar*.

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

3. Para extrair arquivos do arquivo jar, você pode executar o seguinte comando:

```
$ jar xf acme-android-appium-1.0-SNAPSHOT-tests.jar
```

4. Assim que conseguir extrair os arquivos, execute o seguinte comando:

```
$ tree .
```

Você deve encontrar pelo menos uma classe na árvore do diretório de trabalho:

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
everything built from the ./src/test directory)
|- one-class-file.class
```

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

Para obter mais informações, consulte Execute testes Appium automaticamente no 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 Execute testes Appium automaticamente no 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 Execute testes Appium automaticamente no 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 Execute testes Appium automaticamente no 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 Execute testes Appium automaticamente no 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 Execute testes Appium automaticamente no 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 Execute testes Appium automaticamente no 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 Execute testes Appium automaticamente no 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 Execute testes Appium automaticamente no 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 Execute testes Appium automaticamente no Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEELS_INSUFFICIENT

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

> **Warning**
>
> Não foi possível encontrar dependências de wheel suficientes no diretório wheelhouse. Descompacte seu pacote de teste e, em seguida, abra o diretório wheelhouse. Verifique se você tem todas as dependências de wheel especificadas no arquivo requirements.txt.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip test_bundle.zip
```

2. Verifique o tamanho do arquivo *requirements.txt*, bem como o número de arquivos dependentes *.whl* no diretório wheelhouse:

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

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

   - Há um arquivo dependente *.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 *.whl*.

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

Para obter mais informações, consulte Execute testes Appium automaticamente no Device Farm.

Solução de problemas de testes de aplicações Web Appium Python no AWS Device Farm

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

APPIUM_WEB_PYTHON_TEST_PACKAGE_UNZIP_FAILED

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

> **Warning**
>
> Não conseguimos abrir o arquivo de teste ZIP do Appium. Verifique se o arquivo é válido e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip test_bundle.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Um pacote válido do Appium Python deve gerar um resultado semelhante ao seguinte:

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

Para obter mais informações, consulte Execute testes Appium automaticamente no Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING

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

> **Warning**
>
> Não foi possível localizar um arquivo de dependência wheel na árvore do diretório wheelhouse. Descompacte o pacote de testes e abra o diretório wheelhouse, verifique se pelo menos um arquivo wheel encontra-se no diretório e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip test_bundle.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o pacote Appium Python for válido, você encontrará pelo menos um arquivo *.whl* dependente, como os arquivos destacados, dentro do diretório. *wheelhouse*

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

Para obter mais informações, consulte Execute testes Appium automaticamente no Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PLATFORM

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

> **Warning**
>
> Encontramos pelo menos um arquivo wheel que especificou uma plataforma para a qual não oferecemos compatibilidade. Descompacte seu pacote de testes e abra o diretório wheelhouse, verifique se os nomes dos arquivos wheel terminam com -any.whl ou -linux_x86_64.whl e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip test_bundle.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o pacote Appium Python for válido, você encontrará pelo menos um arquivo *.whl* dependente, como os arquivos destacados, dentro do diretório. *wheelhouse* O nome do arquivo pode ser diferente, mas deve terminar com *-any.whl* ou*-linux_x86_64.whl*, que especifica a plataforma. Não há suporte para outras plataformas como `windows`.

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

Para obter mais informações, consulte Execute testes Appium automaticamente no Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING

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

> **Warning**
>
> Não foi possível encontrar o diretório tests em seu pacote de testes. Descompacte o pacote de testes, verifique se o diretório tests está no pacote e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip test_bundle.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o pacote Appium Python for válido, você encontrará o *tests* diretório dentro do diretório de trabalho.

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

Para obter mais informações, consulte Execute testes Appium automaticamente no Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME

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

> **Warning**
>
> Não foi possível localizar um arquivo de teste válido na árvore do diretório tests. Descompacte seu pacote de testes e abra o diretório tests, verifique se pelo menos um nome de arquivo começa ou termina com a palavra-chave "test" e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip test_bundle.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o pacote Appium Python for válido, você encontrará o *tests* diretório dentro do diretório de trabalho. O nome do arquivo pode ser diferente, mas deve começar com *test_* ou terminar com*_test.py*.

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

Para obter mais informações, consulte Execute testes Appium automaticamente no Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISS

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

> **Warning**
>
> Não foi possível encontrar o arquivo requirements.txt em seu pacote de testes. Descompacte o pacote de testes, verifique se o arquivo requirements.txt encontra-se no pacote e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip test_bundle.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Se o pacote Appium Python for válido, você encontrará o *`requirements.txt`* arquivo dentro do diretório de trabalho.

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

   Para obter mais informações, consulte Execute testes Appium automaticamente no Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION

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

> **Warning**
>
> Percebemos que a versão pytest era anterior à versão mínima compatível 2.8.0. Altere a versão pytest no arquivo requirements.txt e tente novamente.

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é test_bundle.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip test_bundle.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Você deve encontrar o *`requirement.txt`* arquivo dentro do diretório de trabalho.

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

3. Para obter a versão pytest, você pode executar o seguinte comando:

   ```
   $ grep "pytest" requirements.txt
   ```

Você provavelmente chegará a um resultado como o seguinte:

```
pytest==2.9.0
```

Ele mostra a versão do pytest que, neste exemplo, é 2.9.0. Se o pacote do Appium Python for válido, a versão de pytest deverá ser posterior ou igual a 2.8.0.

Para obter mais informações, consulte Execute testes Appium automaticamente no 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 Execute testes Appium automaticamente no 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 Execute testes Appium automaticamente no Device Farm.

Solução de problemas de testes de instrumentação no AWS Device Farm

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

> **Note**
>
> Para analisar considerações importantes sobre testes de instrumentação no AWS Device Farm, consulte Instrumentação para Android e AWS Device Farm.

INSTRUMENTATION_TEST_PACKAGE_UNZIP_FAILED

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

```
Warning: We could not open your test APK file. Please verify that the file is valid and
    try again.
```

Verifique se você consegue descompactar o pacote de testes sem erros. No exemplo a seguir, o nome do pacote é app-debug-androidTest-unaligned.apk.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip app-debug-androidTest-unaligned.apk
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Um pacote de testes de instrumentação válido deve gerar um resultado semelhante ao seguinte:

   ```
   .
   ```

```
|-- AndroidManifest.xml
|--  classes.dex
|-- resources.arsc
|-- LICENSE-junit.txt
|-- junit (directory)
`-- META-INF (directory)
```

Para obter mais informações, consulte Instrumentação para Android e AWS Device Farm.

INSTRUMENTATION_TEST_PACKAGE_AAPT_DEBUG_BADGING_FAILED

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

```
We could not extract information about your test package. Please verify that the
      test package is valid by running the command "aapt debug badging <path to your
 test
      package>", and try again after the command does not print any error.
```

Durante o processo de validação do upload, o Device Farm analisa as informações da saída do comando `aapt debug badging <path to your package>`.

Verifique se você consegue executar esse comando com êxito em seu pacote de testes de instrumentação.

No exemplo a seguir, o nome do pacote é app-debug-androidTest-unaligned.apk.

- Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

  ```
  $ aapt debug badging app-debug-androidTest-unaligned.apk
  ```

  Um pacote de testes de instrumentação válido deve gerar um resultado semelhante ao seguinte:

  ```
  package: name='com.amazon.aws.adf.android.referenceapp.test' versionCode=''
   versionName='' platformBuildVersionName='5.1.1-1819727'
  sdkVersion:'9'
  targetSdkVersion:'22'
  application-label:'Test-api'
  application: label='Test-api' icon=''
  application-debuggable
  uses-library:'android.test.runner'
  ```

```
feature-group: label=''
uses-feature: name='android.hardware.touchscreen'
uses-implied-feature: name='android.hardware.touchscreen' reason='default feature
 for all apps'
supports-screens: 'small' 'normal' 'large' 'xlarge'
supports-any-density: 'true'
locales: '--_--'
densities: '160'
```

Para obter mais informações, consulte Instrumentação para Android e AWS Device Farm.

INSTRUMENTATION_TEST_PACKAGE_INSTRUMENTATION_RUNNER_VALU

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

```
We could not find the instrumentation runner value in the AndroidManifest.xml.
      Please verify the test package is valid by running the command "aapt dump xmltree
 <path to
      your test package> AndroidManifest.xml", and try again after finding the
 instrumentation
      runner value behind the keyword "instrumentation."
```

Durante o processo de validação de upload, o Device Farm analisa o valor do executor de instrumentação da árvore de análise XML para um arquivo XML contido no pacote. É possível usar o seguinte comando da : .: `aapt dump xmltree <path to your package> AndroidManifest.xml`.

Verifique se você consegue executar esse comando com êxito em seu pacote de testes de instrumentação e encontrar o valor de instrumentação.

No exemplo a seguir, o nome do pacote é app-debug-androidTest-unaligned.apk.

- Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

  ```
  $ aapt dump xmltree app-debug-androidTest-unaligned.apk AndroidManifest.xml | grep
   -A5 "instrumentation"
  ```

  Um pacote de testes de instrumentação válido deve gerar um resultado semelhante ao seguinte:

  ```
  E: instrumentation (line=9)
  ```

```
      A: android:label(0x01010001)="Tests for
 com.amazon.aws.adf.android.referenceapp" (Raw: "Tests for
 com.amazon.aws.adf.android.referenceapp")
      A:
 android:name(0x01010003)="android.support.test.runner.AndroidJUnitRunner" (Raw:
 "android.support.test.runner.AndroidJUnitRunner")
      A:
 android:targetPackage(0x01010021)="com.amazon.aws.adf.android.referenceapp" (Raw:
 "com.amazon.aws.adf.android.referenceapp")
      A: android:handleProfiling(0x01010022)=(type 0x12)0x0
      A: android:functionalTest(0x01010023)=(type 0x12)0x0
```

Para obter mais informações, consulte Instrumentação para Android e AWS Device Farm.

INSTRUMENTATION_TEST_PACKAGE_AAPT_DUMP_XMLTREE_FAILED

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

```
We could not find the valid AndroidManifest.xml in your test package. Please
      verify that the test package is valid by running the command "aapt dump xmltree
 <path to
      your test package> AndroidManifest.xml", and try again after the command does not
 print any
      error.
```

Durante o processo de validação de upload, o Device Farm analisa as informações da árvore de análise XML de um arquivo XML contido no pacote usando o seguinte comando: `aapt dump xmltree <path to your package> AndroidManifest.xml`.

Verifique se você consegue executar esse comando com êxito em seu pacote de testes de instrumentação.

No exemplo a seguir, o nome do pacote é app-debug-androidTest-unaligned.apk.

- Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

  ```
  $ aapt dump xmltree app-debug-androidTest-unaligned.apk AndroidManifest.xml
  ```

  Um pacote de testes de instrumentação válido deve gerar um resultado semelhante ao seguinte:

  ```
  N: android=http://schemas.android.com/apk/res/android
  ```

```
 E: manifest (line=2)
   A: package="com.amazon.aws.adf.android.referenceapp.test" (Raw:
"com.amazon.aws.adf.android.referenceapp.test")
   A: platformBuildVersionCode=(type 0x10)0x16 (Raw: "22")
   A: platformBuildVersionName="5.1.1-1819727" (Raw: "5.1.1-1819727")
   E: uses-sdk (line=5)
     A: android:minSdkVersion(0x0101020c)=(type 0x10)0x9
     A: android:targetSdkVersion(0x01010270)=(type 0x10)0x16
   E: instrumentation (line=9)
     A: android:label(0x01010001)="Tests for
com.amazon.aws.adf.android.referenceapp" (Raw: "Tests for
com.amazon.aws.adf.android.referenceapp")
     A:
android:name(0x01010003)="android.support.test.runner.AndroidJUnitRunner" (Raw:
"android.support.test.runner.AndroidJUnitRunner")
     A:
android:targetPackage(0x01010021)="com.amazon.aws.adf.android.referenceapp" (Raw:
"com.amazon.aws.adf.android.referenceapp")
     A: android:handleProfiling(0x01010022)=(type 0x12)0x0
     A: android:functionalTest(0x01010023)=(type 0x12)0x0
   E: application (line=16)
     A: android:label(0x01010001)=@0x7f020000
     A: android:debuggable(0x0101000f)=(type 0x12)0xffffffff
     E: uses-library (line=17)
       A: android:name(0x01010003)="android.test.runner" (Raw:
"android.test.runner")
```

Para obter mais informações, consulte Instrumentação para Android e AWS Device Farm.

INSTRUMENTATION_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISS

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

```
We could not find the package name in your test package. Please verify that the
      test package is valid by running the command "aapt debug badging <path to your
 test
      package>", and try again after finding the package name value behind the keyword
 "package:
      name."
```

Durante o processo de validação do upload, o Device Farm analisa o valor do nome do pacote pela saída do seguinte comando: `aapt debug badging <path to your package>`.

Verifique se você consegue executar esse comando com êxito em seu pacote de testes de instrumentação e encontrar o valor do nome do pacote.

No exemplo a seguir, o nome do pacote é app-debug-androidTest-unaligned.apk.

- Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

  ```
  $ aapt debug badging app-debug-androidTest-unaligned.apk | grep "package: name="
  ```

  Um pacote de testes de instrumentação válido deve gerar um resultado semelhante ao seguinte:

  ```
  package: name='com.amazon.aws.adf.android.referenceapp.test' versionCode=''
   versionName='' platformBuildVersionName='5.1.1-1819727'
  ```

  Para obter mais informações, consulte Instrumentação para Android e AWS Device Farm.

Solução de problemas de testes de aplicações iOS no AWS Device Farm

O tópico a seguir lista mensagens de erro que ocorrem durante o upload de testes de aplicativos iOS e recomenda soluções para resolver cada erro.

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

IOS_APP_UNZIP_FAILED

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

> **Warning**
>
> Não conseguimos abrir seu aplicativo. Verifique se o arquivo é válido e tente novamente.

Verifique se você consegue descompactar o pacote de aplicativos sem erros. No exemplo a seguir, o nome do pacote é 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 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 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 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 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 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 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 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 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 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 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 Integrando o Device Farm com XCTest o iOS.

XCTEST_TEST_PACKAGE_XCTEST_DIR_MISSING

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

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

No exemplo a seguir, o nome do pacote é swiftExampleTests.xctest-1.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swiftExampleTests.xctest-1.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Se o 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 Integrando o Device Farm com XCTest o iOS.

XCTEST_TEST_PACKAGE_PLIST_FILE_MISSING

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

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

No exemplo a seguir, o nome do pacote é swiftExampleTests.xctest-1.zip.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Se o 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 Integrando o Device Farm com XCTest o iOS.

XCTEST_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING

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

> **Warning**
>
> Não foi possível encontrar o valor do nome do pacote no arquivo Info.plist. Descompacte 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 Integrando o Device Farm com XCTest o iOS.

XCTEST_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING

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

> **Warning**
>
> Não foi possível encontrar o valor do executável no arquivo Info.plist. Descompacte o pacote de 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 Integrando o Device Farm com XCTest o iOS.

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.

```
We could not open your test IPA file. Please verify that the file is valid and try again.
```

Verifique se você consegue descompactar o pacote de aplicativos sem erros. No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Um pacote de aplicativos iOS válido deve gerar um resultado semelhante ao seguinte:

   ```
   .
   ```

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

Para obter mais informações, consulte Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_MISSING

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

```
We could not find the Payload directory inside your test package. Please unzip your test package, verify that the Payload directory is inside the package, and try again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

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

XCTEST_UI_TEST_PACKAGE_APP_DIR_MISSING

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

```
We could not find the .app directory inside the Payload directory. Please unzip your test package and then open the Payload directory, verify that the .app directory is inside the directory, and try again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip swift-sample-UI.ipa
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

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

XCTEST_UI_TEST_PACKAGE_PLUGINS_DIR_MISSING

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

```
We could not find the Plugins directory inside the .app directory. Please unzip your test package and then open the .app directory, verify that the Plugins directory is inside the directory, and try again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

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

XCTEST_UI_TEST_PACKAGE_XCTEST_DIR_MISSING_IN_PLUGINS_DIR

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

```
We could not find the .xctest directory inside the plugins directory.
Please unzip your test package and then open the plugins directory, verify
that the .xctest directory is inside the directory, and try again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

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

XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING

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

```
We could not find the Info.plist file inside the .app directory. Please
unzip your test package and then open the .app directory, verify that the
Info.plist file is inside the directory, and try again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

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

XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING_IN_XCTEST_DIR

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

```
We could not find the Info.plist file inside the .xctest directory. Please
unzip your test package and then open the .xctest directory, verify that
the Info.plist file is inside the directory, and try again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip swift-sample-UI.ipa
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

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

XCTEST_UI_TEST_PACKAGE_CPU_ARCHITECTURE_VALUE_MISSING

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

```
We could not the CPU architecture value in the Info.plist file. Please
unzip your test package and then open the Info.plist file inside the .app
directory, verify that the key "UIRequiredDeviceCapabilities" is specified,
and try again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip swift-sample-UI.ipa
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Você deve encontrar o 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 Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_PLATFORM_VALUE_MISSING

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

```
We could not find the platform value in the Info.plist. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the key "CFBundleSupportedPlatforms" is specified, and try again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Você deve encontrar o 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 Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_WRONG_PLATFORM_DEVICE_VALUE

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

```
We found the platform device value was wrong in the Info.plist file. Please
unzip your test package and then open the Info.plist file inside the .app
directory, verify that the value of the key "CFBundleSupportedPlatforms"
does not contain the keyword "simulator", and try again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip swift-sample-UI.ipa
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Você deve encontrar o 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 Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_FORM_FACTOR_VALUE_MISSING

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

`We could not the form factor value in the Info.plist. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the key "UIDeviceFamily" is specified, and try again.`

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

```
$ unzip swift-sample-UI.ipa
```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Você deve encontrar o 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 Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING

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

```
We could not find the package name value in the Info.plist file. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the key "CFBundleIdentifier" is specified, and try again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Você deve encontrar o 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 Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING

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

```
We could not find the executable value in the Info.plist file. Please
unzip your test package and then open the Info.plist file inside the .app
directory, verify that the key "CFBundleExecutable" is specified, and try
again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Você deve encontrar o 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 Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSING

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

```
We could not find the package name value in the Info.plist file inside
the .xctest directory. Please unzip your test package and then open
the Info.plist file inside the .xctest directory, verify that the key
"CFBundleIdentifier" is specified, and try again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

```
$ tree .
```

Você deve encontrar o 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 Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_TEST_EXECUTABLE_VALUE_MISSING

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

```
We could not find the executable value in the Info.plist file inside
the .xctest directory. Please unzip your test package and then open
the Info.plist file inside the .xctest directory, verify that the key
"CFBundleExecutable" is specified, and try again.
```

No exemplo a seguir, o nome do pacote é swift-sample-UI.ipa.

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Você deve encontrar o 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 Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_MULTIPLE_APP_DIRS

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

`We found multiple .app directories inside your test package. Please unzip your test package, verify that only a single .app directory is present inside the package, then try again.`

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swift-sample-UI.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Se o pacote XCTest UI for válido, você deverá encontrar somente um único diretório `.app`, como `swift-sampleUITests-Runner.app`, no nosso exemplo, no pacote de teste .zip.

   ```
   .
   `--swift-sample-UI.zip--(directory)
       `-- swift-sampleUITests-Runner.app (directory)
               |-- Info.plist
               |-- Plugins (directory)
               |       `swift-sampleUITests.xctest (directory)
   ```

```
          |            |-- Info.plist
          |            `-- (any other files)
          `-- (any other files)
    `-- (any other files)
```

Para obter mais informações, consulte Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_MULTIPLE_IPA_DIRS

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

`We found multiple .ipa directories inside your test package. Please unzip your test package, verify that only a single .ipa directory is present inside the package, then try again.`

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swift-sample-UI.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Se o pacote XCTest UI for válido, você deverá encontrar somente um único diretório `.ipa`, como `sampleUITests.ipa`, no nosso exemplo, no pacote de teste .zip.

   ```
   .
   `--swift-sample-UI.zip--(directory)
       `-- sampleUITests.ipa (directory)
               `-- Payload (directory)
                   `-- swift-sampleUITests-Runner.app (directory)
       `-- (any other files)
   ```

   Para obter mais informações, consulte Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_BOTH_APP_AND_IPA_DIR_PRESENT

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

```
We found both .app and .ipa files inside your test package. Please unzip your test package, verify that only a single .app or .ipa file is present inside the package, then try again.
```

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swift-sample-UI.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Se o pacote XCTest UI for válido, você deverá encontrar um diretório `.ipa`, como `sampleUITests.ipa`, ou `.app`, como `swift-sampleUITests-Runner.app` em nosso exemplo, no pacote de teste .zip. É possível consultar um exemplo de pacote de teste XCTEST_UI válido em nossa documentação em Integrando a XCTest interface do usuário para iOS com o Device Farm.

   ```
   .
   `--swift-sample-UI.zip--(directory)
       `-- sampleUITests.ipa (directory)
               `-- Payload (directory)
                   `-- swift-sampleUITests-Runner.app (directory)
      `-- (any other files)
   ```

   ou

   ```
   .
   `--swift-sample-UI.zip--(directory)
       `-- swift-sampleUITests-Runner.app (directory)
               |-- Info.plist
               |-- Plugins (directory)
               `-- (any other files)
      `-- (any other files)
   ```

Para obter mais informações, consulte Integrando a XCTest interface do usuário para iOS com o Device Farm.

XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_PRESENT_IN_ZIP

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

```
We found a Payload directory inside your .zip test package. Please unzip your test package, ensure that a Payload directory is not present in the package, then try again.
```

1. Copie o pacote de testes para seu diretório de trabalho e execute o comando a seguir:

   ```
   $ unzip swift-sample-UI.zip
   ```

2. Assim que conseguir descompactar o pacote, você poderá encontrar a estrutura de árvore do diretório de trabalho executando o seguinte comando:

   ```
   $ tree .
   ```

   Se o pacote XCTest UI for válido, você não deverá encontrar um diretório de carga útil dentro do pacote de teste.

   ```
   .
   `--swift-sample-UI.zip--(directory)
       `-- swift-sampleUITests-Runner.app (directory)
               |-- Info.plist
               |-- Plugins (directory)
               `-- (any other files)
      `-- Payload (directory) [This directory should not be present]
               |-- (any other files)
      `-- (any other files)
   ```

   Para obter mais informações, consulte Integrando a XCTest interface do usuário para iOS com o Device Farm.

Segurança em AWS Device Farm

A segurança na nuvem AWS é a maior prioridade. Como AWS cliente, você se beneficia de uma arquitetura de data center e rede criada para atender aos requisitos das organizações mais sensíveis à segurança.

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

- Segurança da nuvem — AWS é responsável por proteger a infraestrutura que executa AWS os serviços na AWS nuvem. AWS também fornece serviços que você pode usar com segurança. Auditores terceirizados testam e verificam regularmente a eficácia de nossa segurança como parte dos Programas de Conformidade Programas de AWS de . Para saber mais sobre os programas de conformidade que se aplicam AWS Device Farm, consulte Serviços da AWS no escopo do programa de conformidade .
- Segurança na nuvem: sua responsabilidade é determinada pelo serviço da AWS que você usa. Você também é responsável por outros fatores, incluindo a confidencialidade de seus dados, os requisitos da empresa e as leis e regulamentos aplicáveis.

Esta documentação o ajuda a entender como aplicar o modelo de responsabilidade compartilhada ao usar o Device Farm. Os tópicos a seguir mostram como configurar o Device Farm para atender aos seus objetivos de segurança e conformidade. Você também aprenderá a usar outros serviços da AWS que o ajudam a monitorar e proteger os recursos do Device Farm.

Tópicos

Gerenciamento de identidade e acesso no AWS Device Farm

Público

A forma como você usa AWS Identity and Access Management (IAM) difere com base na sua função:

- Usuário do serviço: solicite permissões ao seu administrador se você não conseguir acessar os atributos (consulte Solução de problemas de identidade e acesso ao AWS Device Farm).
- Administrador do serviço: determine o acesso do usuário e envie solicitações de permissão (consulte Como o AWS Device Farm funciona com o IAM)
- Administrador do IAM: escreva políticas para gerenciar o acesso (consulte Exemplos de políticas baseadas em identidade do AWS Device Farm)

Autenticação com identidades

A autenticação é como você faz login AWS usando suas credenciais de identidade. Você deve estar autenticado como usuário do IAM ou assumindo uma função do IAM. Usuário raiz da conta da AWS

Você pode fazer login como uma identidade federada usando credenciais de uma fonte de identidade como Centro de Identidade do AWS IAM (IAM Identity Center), autenticação de login único ou credenciais. Google/Facebook Para ter mais informações sobre como fazer login, consulte Como fazer login em sua Conta da AWS no Guia do usuário do Início de Sessão da AWS .

Para acesso programático, AWS fornece um SDK e uma CLI para assinar solicitações criptograficamente. Para ter mais informações, consulte AWS Signature Version 4 para solicitações de API no Guia do usuário do IAM.

Conta da AWS usuário root

Ao criar um Conta da AWS, você começa com uma identidade de login chamada usuário Conta da AWS raiz que tem acesso completo a todos Serviços da AWS os recursos. É altamente recomendável não usar o usuário-raiz em tarefas diárias. Consulte as tarefas que exigem credenciais de usuário-raiz em 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 com permissões específicas para uma única pessoa ou aplicação. É recomendável usar credenciais temporárias, em vez de usuários do IAM com

credenciais de longo prazo. Para obter mais informações, consulte Exigir que usuários humanos usem a federação com um provedor de identidade para acessar AWS usando credenciais temporárias no Guia do usuário do IAM.

Um grupo do IAM especifica um conjunto de usuários do IAM e facilita o gerenciamento de permissões para grandes conjuntos de usuários. Para ter mais informações, consulte Casos de uso de usuários do IAM no Guia do usuário do IAM.

Perfis do IAM

Uma perfil do IAM é uma identidade com permissões específicas que oferece credenciais temporárias. Você pode assumir uma função mudando de um usuário para uma função do IAM (console) ou chamando uma operação de AWS API AWS CLI ou. Para saber mais, consulte Métodos para assumir um perfil no Manual do usuário do IAM.

Os perfis do IAM são úteis para acesso de usuário federado, permissões de usuário do IAM temporárias, acesso entre contas, acesso entre serviços e aplicações em execução no Amazon EC2. Consulte mais informações em Acesso a recursos entre contas no IAM no Guia do usuário do IAM.

Como o AWS Device Farm funciona com o IAM

Antes de usar o IAM para gerenciar o acesso ao Device Farm, você deve entender quais recursos do IAM estão disponíveis para uso com o Device Farm. Para ter uma visão geral de como o Device Farm e outros AWS serviços funcionam com o IAM, consulte AWS Services That Work with IAM no Guia do usuário do IAM.

Tópicos

- Políticas baseadas em identidade do Device Farm
- Políticas baseadas em recursos do Device Farm
- Listas de controle de acesso
- Autorização baseada em tags do Device Farm
- Perfis do IAM do Device Farm

Políticas baseadas em identidade do Device Farm

Com as políticas baseadas em identidade do IAM, é possível especificar ações ou recursos permitidos ou negados, além das condições sob as quais as ações são permitidas ou negadas. O Device Farm oferece suporte a ações, recursos e chaves de condição específicos. Para conhecer

todos os elementos usados em uma política JSON, consulte Referência de elementos de política JSON do IAM no Guia do usuário do IAM.

Ações

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

O elemento `Action` de uma política JSON descreve as ações que podem ser usadas para permitir ou negar acesso em uma política. Incluem ações em uma política para conceder permissões para executar a operação associada.

As ações de política no Device Farm usam o seguinte prefixo antes da ação: `devicefarm:`. Por exemplo, para conceder permissão a alguém para iniciar sessões do Selenium com a operação de API `CreateTestGridUrl` de teste de navegador de desktop do Device Farm, você inclui a ação `devicefarm:CreateTestGridUrl` na política. As instruções de política devem incluir um elemento `Action` ou `NotAction`. O Device Farm define seu próprio conjunto de ações que descrevem as tarefas que você pode executar com esse serviço.

Para especificar várias ações em uma única instrução, separe-as com vírgulas, como segue:

```
"Action": [
      "devicefarm:action1",
      "devicefarm:action2"
```

Você também pode especificar várias ações usando caracteres curinga (*). Por exemplo, para especificar todas as ações que começam com a palavra `List`, inclua a seguinte ação:

```
"Action": "devicefarm:List*"
```

Para ver uma lista de ações do Device Farm, consulte Actions defined by AWS Device Farm na Referência de autorização de serviço do IAM.

Recursos

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

O elemento de política JSON `Resource` especifica o objeto ou os objetos aos quais a ação se aplica. Como prática recomendada, especifique um recurso usando seu nome do recurso da Amazon

(ARN). Para ações que não oferecem compatibilidade com permissões em nível de recurso, use um 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 Amazon Resource Names (ARNs) e AWS Service Namespaces.

Por exemplo, para especificar a instância `i-1234567890abcdef0` na instrução, use o seguinte ARN:

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:instance/i-1234567890abcdef0"
```

Para especificar todas as instâncias que pertencem a uma conta, use o caractere curinga (*):

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:instance/*"
```

Algumas ações do Device Farm, como as de criação de recursos, não podem ser executadas em um recurso. Nesses casos, é necessário 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 AWS Device Farm na Referência de autorização de serviço do IAM. Para saber com quais

ações você pode especificar o ARN de cada recurso, consulte Actions defined by AWS Device Farm na Referência de autorização de serviço do IAM.

Chaves de condição

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

O elemento `Condition` especifica quando as instruções são executadas com base em critérios definidos. É possível criar expressões condicionais que usem agentes de condição, como “igual a” ou “menor que”, para fazer a condição da política corresponder aos valores na solicitação. Para ver todas as chaves de condição AWS globais, consulte as chaves de contexto de condição AWS global no Guia do usuário do IAM.

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

Para ver uma lista das chaves de condição do Device Farm, consulte Condition keys for AWS Device Farm na Referência de autorização de serviço do IAM. Para saber com quais ações e recursos você pode usar uma chave de condição, consulte Actions defined by AWS Device Farm na Referência de autorização de serviço do IAM.

Exemplos

Para ver exemplos de políticas baseadas em identidade do Device Farm, consulte Exemplos de políticas baseadas em identidade do AWS Device Farm.

Políticas baseadas em recursos do Device Farm

O Device Farm não é compatível com políticas baseadas em recursos.

Listas de controle de acesso

O Device Farm não suporta listas de controle de acesso (ACLs).

Autorização baseada em tags do Device Farm

Você pode anexar tags aos recursos do Device Farm ou passar tags em uma solicitação para o Device Farm. Para controlar o acesso baseado em tags, forneça informações sobre as

tags no elemento de condição de uma política usando as `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` ou chaves de condição `aws:TagKeys`. Para obter mais informações sobre a marcação de recursos do Device Farm, consulte Marcação no Device Farm.

Para visualizar um exemplo de política baseada em identidade para limitar o acesso a um recurso baseado em tags desse recurso, consulte Visualizando projetos de teste do navegador de desktop Device Farm com base em tags.

Perfis do IAM do Device Farm

Uma função do IAM é uma entidade na sua AWS conta que tem permissões específicas.

Uso de credenciais temporárias com o Device Farm

O Device Farm é compatível com o uso de credenciais temporárias.

Você pode usar credenciais temporárias para fazer login com a federação e assumir um perfil do IAM ou um perfil entre contas. Você obtém credenciais de segurança temporárias chamando operações de AWS STS API, como AssumeRoleou GetFederationToken.

Perfis vinculados ao serviço

As funções vinculadas ao serviço permitem que AWS os serviços acessem recursos em outros serviços para concluir uma ação em seu nome. Os perfis vinculados a serviço aparecem em sua conta do IAM e são de propriedade do serviço. Um administrador do IAM pode visualizar, mas não pode editar, as permissões das funções vinculadas a serviços.

O Device Farm usa perfis vinculados ao serviço no recurso de teste do navegador de desktop do Device Farm. Para obter informações sobre esses perfis, consulte Using Service-Linked Roles in Device Farm desktop browser testing no guia do desenvolvedor.

Perfis de serviço

O Device Farm não é compatível com perfis de serviço.

Esse atributo permite que um serviço assuma um perfil de serviço em seu nome. O perfil permite que o serviço acesse recursos em outros serviços para concluir uma ação em seu nome. Os perfis de serviço aparecem em sua conta do IAM e são de propriedade da conta. Isso significa que um administrador do IAM pode alterar as permissões para esse perfil. Porém, fazer isso pode alterar a funcionalidade do serviço.

Gerenciar o acesso usando políticas

Você controla o acesso AWS criando políticas e anexando-as a AWS identidades ou recursos. Uma política define permissões quando associada a uma identidade ou recurso. AWS avalia essas políticas quando um diretor faz uma solicitação. A maioria das políticas é armazenada AWS como documentos JSON. Para ter mais informações sobre documentos de política JSON, consulte Visão geral das políticas JSON no Guia do usuário do IAM.

Por meio de políticas, os administradores especificam quem tem acesso a que, definindo qual entidade principal pode realizar ações em quais recursos e sob quais condições.

Por padrão, usuários e perfis não têm permissões. Um administrador do IAM cria políticas do IAM e as adiciona aos perfis, os quais os usuários podem então assumir. As políticas do IAM definem permissões, independentemente do método usado para realizar a operação.

Políticas baseadas em identidade

As políticas baseadas em identidade são documentos de políticas de permissão JSON que você anexa a uma identidade (usuário, grupo ou perfil). Essas políticas controlam quais ações as identidades podem realizar, em quais recursos e sob quais condições. Para saber como criar uma política baseada em identidade, consulte Definir permissões personalizadas do IAM com as políticas gerenciadas pelo cliente no Guia do Usuário do IAM.

As políticas baseadas em identidade podem ser políticas em linha (incorporadas diretamente em uma única identidade) ou políticas gerenciadas (políticas autônomas anexadas a várias identidades). Para saber como escolher entre uma política gerenciada e políticas em linha, consulte Escolher entre políticas gerenciadas e políticas em linha no Guia do usuário do IAM.

A tabela a seguir descreve as políticas gerenciadas pela AWS do Device Farm.

Alteração	Descrição	Data
AWSDeviceFarmFullAccess	Fornece acesso total a todas as operações do AWS Device Farm.	15 de julho de 2015
AWSServiceRoleForDeviceFarmTestGrid	Permite que o Device Farm acesse recursos da AWS em seu nome.	20 de maio de 2021

Outros tipos de política

AWS oferece suporte a tipos de políticas adicionais que podem definir o máximo de permissões concedidas por tipos de políticas mais comuns:

- Limites de permissões: definem o número máximo de permissões que uma política baseada em identidade pode conceder a uma entidade do IAM. Para saber mais sobre limites de permissões, consulte Limites de permissões para identidades do IAM no Guia do usuário do IAM.
- Políticas de controle de serviço (SCPs) — Especifique as permissões máximas para uma organização ou unidade organizacional em AWS Organizations. Para saber mais, consulte Políticas de controle de serviço no Guia do usuário do AWS Organizations .
- Políticas de controle de recursos (RCPs) — Defina o máximo de permissões disponíveis para recursos em suas contas. Para obter mais informações, consulte Políticas de controle de recursos (RCPs) no Guia AWS Organizations do usuário.
- Políticas de sessão: políticas avançadas transmitidas como um parâmetro durante a criação de uma sessão temporária para um perfil ou um usuário federado. Para saber mais, consulte Políticas de sessão no Guia do usuário do IAM.

Vários tipos de política

Quando vários tipos de política são aplicáveis a uma solicitação, é mais complicado compreender as permissões resultantes. Para saber como AWS determinar se uma solicitação deve ser permitida quando vários tipos de políticas estão envolvidos, consulte Lógica de avaliação de políticas no Guia do usuário do IAM.

Exemplos de políticas baseadas em identidade do AWS Device Farm

Por padrão, os perfis e usuários do IAM não têm permissão para criar ou modificar recursos do Device Farm. Eles também não podem realizar tarefas usando a AWS API Console de gerenciamento da AWS AWS CLI, ou. Um administrador do IAM deve criar políticas do IAM que concedam aos usuários e perfis permissão para executarem operações de API específicas nos recursos especificados de que precisam. O administrador deve anexar essas políticas aos usuários ou grupos do IAM que exigem essas permissões.

Para saber como criar uma política baseada em identidade do IAM usando esses exemplos de documentos de política JSON, consulte Criar políticas na guia JSON no Guia do usuário do IAM.

Tópicos

- Práticas recomendadas de política
- Permitir que os usuários visualizem suas próprias permissões
- Acesso a um projeto de teste de navegador de desktop do Device Farm
- Visualizando projetos de teste do navegador de desktop Device Farm com base em tags

Práticas recomendadas de política

As políticas baseadas em identidade determinam se alguém pode criar, acessar ou excluir recursos do Device Farm em sua conta. Essas ações podem incorrer em custos para sua Conta da AWS. Ao criar ou editar políticas baseadas em identidade, siga estas diretrizes e recomendações:

- Comece com as políticas AWS gerenciadas e avance para as permissões de privilégios mínimos — Para começar a conceder permissões aos seus usuários e cargas de trabalho, use as políticas AWS gerenciadas que concedem permissões para muitos casos de uso comuns. Eles estão disponíveis no seu Conta da AWS. Recomendamos que você reduza ainda mais as permissões definindo políticas gerenciadas pelo AWS cliente que sejam específicas para seus casos de uso. Para saber mais, 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 saber mais 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: é possível adicionar uma condição às políticas para limitar o acesso a ações e recursos. Por exemplo, é possível escrever uma condição de política para especificar que todas as solicitações devem ser enviadas usando SSL. Você também pode usar condições para conceder acesso às ações de serviço se elas forem usadas por meio de uma ação específica AWS service (Serviço da AWS), como CloudFormation. Para saber mais, consulte Elementos da política JSON do IAM: condição no Guia do usuário do IAM.
- Use o IAM Access Analyzer para validar suas políticas do IAM a fim de garantir permissões seguras e funcionais: o IAM Access Analyzer valida as políticas novas e existentes para que elas sigam a linguagem de política do IAM (JSON) e as práticas recomendadas do IAM. O IAM Access Analyzer oferece mais de cem verificações de política e recomendações práticas para ajudar a criar políticas seguras e funcionais. Para saber mais, consulte Validação de políticas do IAM Access Analyzer no Guia do Usuário do IAM.

- Exigir autenticação multifator (MFA) — Se você tiver um cenário que exija usuários do IAM ou um usuário root, ative Conta da AWS a MFA para obter segurança adicional. Para exigir MFA quando as operações de API forem chamadas, adicione condições de MFA às suas políticas. Para saber mais, consulte Configuração de acesso à API protegido por MFA no Guia do Usuário do IAM.

Para saber mais sobre as práticas recomendadas do IAM, consulte Práticas recomendadas de segurança no IAM no Guia do usuário do IAM.

Permitir que os usuários visualizem suas próprias permissões

Este exemplo mostra como criar uma política que permita que os usuários do IAM visualizem as políticas gerenciadas e em linha anexadas a sua identidade de usuário. Essa política inclui permissões para concluir essa ação no console ou programaticamente usando a API AWS CLI ou AWS .

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
```

```
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Acesso a um projeto de teste de navegador de desktop do Device Farm

Neste exemplo, você deseja conceder a um usuário do IAM em sua AWS conta acesso a um dos seus projetos de teste do navegador de desktop Device Farm,. `arn:aws:devicefarm:us-west-2:111122223333:testgrid-project:123e4567-e89b-12d3-a456-426655441111` Você deseja que a conta seja capaz de ver itens relacionados ao projeto.

Além do endpoint `devicefarm:GetTestGridProject`, a conta deve ter os endpoints `devicefarm:ListTestGridSessionArtifacts` `devicefarm:ListTestGridSessions`, `devicefarm:GetTestGridSession` e `devicefarm:ListTestGridSessionActions`.

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:

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.

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 `0wner=richard-roe` ou `owner=richard-roe`. Caso contrário, o usuário terá o acesso negado. A chave da tag de condição `0wner` corresponde a `0wner` e a `owner` porque os nomes de chaves de condição não diferenciam letras maiúsculas de minúsculas. Para obter mais informações, consulte IAM JSON Policy Elements: Condition (Elementos da política JSON do IAM: Condição) no Guia do usuário do IAM.

Solução de problemas de identidade e acesso ao AWS Device Farm

Use as informações a seguir para ajudá-lo a diagnosticar e corrigir problemas comuns que você pode encontrar ao trabalhar com o Device Farm e o IAM.

Não estou autorizado a realizar uma ação no Device Farm

Se você receber uma mensagem de erro Console de gerenciamento da AWS informando que você não está autorizado a realizar uma ação, entre em contato com o administrador para obter ajuda. O administrador é a pessoa que forneceu a você o seu nome de usuário e senha.

O exemplo de erro a seguir ocorre quando o usuário do IAM, `mateojackson`, tenta usar o console para visualizar detalhes sobre uma execução, mas não tem permissões `devicefarm:GetRun`.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 devicefarm:GetRun on resource: arn:aws:devicefarm:us-west-2:123456789101:run:123e4567-
e89b-12d3-a456-426655440000/123e4567-e89b-12d3-a456-426655441111
```

Nesse caso, Mateo pede ao administrador para atualizar suas políticas para que ele tenha acesso ao `devicefarm:GetRun` no recurso `arn:aws:devicefarm:us-west-2:123456789101:run:123e4567-e89b-12d3-a456-426655440000/123e4567-e89b-12d3-a456-426655441111` usando a ação `devicefarm:GetRun`.

Não estou autorizado a realizar iam: PassRole

Se você receber um erro informando que não está autorizado a executar a ação `iam:PassRole`, suas políticas deverão ser atualizadas para permitir que você passe um perfil para o Device Farm.

Alguns Serviços da AWS permitem que você passe uma função existente para esse serviço em vez de criar uma nova função de serviço ou uma função vinculada ao serviço. Para fazer isso, é preciso ter permissões para passar o perfil para o serviço.

O exemplo de erro a seguir ocorre quando um usuário do IAM chamado `marymajor` tenta usar o console para executar uma ação no Device Farm. No entanto, a ação exige que o serviço tenha permissões concedidas por um perfil de serviço. Mary não tem permissões para passar o perfil para o serviço.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

Nesse caso, as políticas de Mary devem ser atualizadas para permitir que ela realize a ação `iam:PassRole`.

Se precisar de ajuda, entre em contato com seu AWS administrador. Seu administrador é a pessoa que forneceu suas credenciais de login.

Quero visualizar minhas chaves de acesso

Depois de criar suas chaves de acesso de usuário do IAM, é possível visualizar seu ID da chave de acesso a qualquer momento. No entanto, você não pode visualizar sua chave de acesso secreta novamente. Se você perder sua chave secreta, crie um novo par de chaves de acesso.

As chaves de acesso consistem em duas partes: um ID de chave de acesso (por exemplo, `AKIAIOSFODNN7EXAMPLE`) e uma chave de acesso secreta (por exemplo, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Como um nome de usuário e uma senha, você deve usar o ID da chave de acesso e a chave de acesso secreta em conjunto para autenticar suas solicitações. Gerencie suas chaves de acesso de forma tão segura quanto você gerencia seu nome de usuário e sua senha.

> **Important**
>
> Não forneça as chaves de acesso a terceiros, mesmo que seja para ajudar a encontrar o ID de usuário canônico. Ao fazer isso, você pode dar a alguém acesso permanente ao seu Conta da AWS.

Ao criar um par de chaves de acesso, você é solicitado a guardar o ID da chave de acesso e a chave de acesso secreta em um local seguro. A chave de acesso secreta só está disponível no momento em que é criada. Se você perder sua chave de acesso secreta, será necessário adicionar novas chaves de acesso para seu usuário do IAM. Você pode ter no máximo duas chaves de acesso. Se você já tiver duas, você deverá excluir um par de chaves para poder criar um novo. Para visualizar as instruções, consulte Gerenciar chaves de acesso no Guia do usuário do IAM.

Sou um administrador e quero permitir que outras pessoas acessem o Device Farm

Para permitir que outras pessoas acessem o Device Farm, você deve conceder permissão às pessoas ou aplicações que precisam de acesso. Se você estiver usando o Centro de Identidade do AWS IAM para gerenciar pessoas e aplicações, atribua conjuntos de permissões a

usuários ou grupos para definir o nível de acesso. Os conjuntos de permissões criam e atribuem automaticamente políticas do IAM aos perfis do IAM associados à pessoa ou aplicação. Para ter mais informações, consulte Conjuntos de permissões no Guia do usuário do Centro de Identidade do AWS IAM .

Se você não estiver usando o Centro de Identidade do IAM, deverá criar entidades do IAM (usuários ou perfis) para as pessoas ou aplicações que precisam de acesso. Em seguida, você deve anexar uma política à entidade que concede a ela as permissões corretas no Device Farm. Depois que as permissões forem concedidas, forneça as credenciais ao usuário ou desenvolvedor da aplicação. Eles usarão essas credenciais para acessar AWS. Para saber mais sobre como criar grupos, políticas, permissões e usuários do IAM, consulte Identidades do IAM e Políticas e permissões no IAM no Guia do usuário do IAM.

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

É possível criar um perfil que os usuários de outras contas ou pessoas fora da organização podem usar para acessar seus recursos. É possível especificar quem é confiável para assumir o perfil. Para serviços que oferecem suporte a políticas baseadas em recursos ou listas de controle de acesso (ACLs), você pode usar essas políticas para conceder às pessoas acesso aos seus recursos.

Para saber mais, consulte:

- Para saber se o Device Farm é compatível com esses recursos, consulte Como o AWS Device Farm funciona com o IAM.
- Para saber como fornecer acesso aos seus recursos em todos os Contas da AWS que você possui, consulte Como fornecer acesso a um usuário do IAM em outro Conta da AWS que você possui no Guia do usuário do IAM.
- Para saber como fornecer acesso aos seus recursos a terceiros Contas da AWS, consulte Como fornecer acesso Contas da AWS a terceiros no Guia do usuário do IAM.
- Para saber como conceder acesso por meio da federação de identidades, consulte Conceder acesso a usuários autenticados externamente (federação de identidades) no Guia do usuário do IAM.
- Para conhecer a diferença entre perfis e políticas baseadas em recurso para acesso entre contas, consulte Acesso a recursos entre contas no IAM no Guia do usuário do IAM.

Validação de conformidade 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 Guia do desenvolvedor AWS Config: AWS Config avalia a conformidade das configurações de seus recursos com práticas internas, diretrizes do setor e regulamentos.
- AWS Security Hub CSPM: esse serviço da AWS fornece uma visão abrangente do estado da segurança na AWS que ajuda verificar a conformidade com os padrões e as práticas recomendadas de segurança do setor.

Proteção de dados em AWS Device Farm

O modelo de responsabilidade AWS compartilhada modelo de se aplica à proteção de dados em AWS Device Farm (Device Farm). Conforme descrito neste modelo, AWS é responsável por proteger a infraestrutura global que executa todos os Nuvem AWS. Você é responsável por manter o controle sobre o conteúdo hospedado nessa infraestrutura. Você também é responsável pelas tarefas de configuração e gerenciamento de segurança dos Serviços da AWS que usa. Para obter mais informações sobre a privacidade de dados, consulte as Data Privacy FAQ. Para obter mais

informações sobre a proteção de dados na Europa, consulte a postagem do blog AWS Shared Responsibility Model and RGPD no Blog de segurança da AWS .

Para fins de proteção de dados, recomendamos que você proteja Conta da AWS as credenciais e configure usuários individuais com Centro de Identidade do AWS IAM ou AWS Identity and Access Management (IAM). Dessa maneira, cada usuário receberá apenas as permissões necessárias para cumprir suas obrigações de trabalho. Recomendamos também que você proteja seus dados das seguintes formas:

- Use uma autenticação multifator (MFA) com cada conta.
- Use SSL/TLS para se comunicar com AWS os recursos. Exigimos TLS 1.2 e recomendamos TLS 1.3.
- Configure a API e o registro de atividades do usuário com AWS CloudTrail. Para obter informações sobre o uso de CloudTrail trilhas para capturar AWS atividades, consulte Como trabalhar com CloudTrail trilhas no Guia AWS CloudTrail do usuário.
- Use soluções de AWS criptografia, juntamente com todos os controles de segurança padrão Serviços da AWS.
- Use serviços gerenciados de segurança avançada, como o Amazon Macie, que ajuda a localizar e proteger dados sensíveis armazenados no Amazon S3.
- Se você precisar de módulos criptográficos validados pelo FIPS 140-3 ao acessar AWS por meio de uma interface de linha de comando ou de uma API, use um endpoint FIPS. Para obter mais informações sobre os endpoints FIPS disponíveis, consulte Federal Information Processing Standard (FIPS) 140-3.

É altamente recomendável que nunca sejam colocadas informações confidenciais ou sensíveis, como endereços de e-mail de clientes, em tags ou campos de formato livre, como um campo Nome. Isso inclui quando você trabalha com o Device Farm ou outro Serviços da AWS usando o console AWS CLI, a API ou AWS SDKs. Quaisquer dados inseridos em tags ou em campos de texto de formato livre usados para nomes podem ser usados para logs de faturamento ou de diagnóstico. Se você fornecer um URL para um servidor externo, recomendemos fortemente que não sejam incluídas informações de credenciais no URL para validar a solicitação a esse servidor.

Criptografia em trânsito

Os endpoints do Device Farm suportam somente HTTPS assinado (SSL/TLS) requests except where otherwise noted. All content retrieved from or placed in Amazon S3 through upload URLs is encrypted

using SSL/TLS. Para obter mais informações sobre como as solicitações HTTPS são registradas AWS, consulte Assinatura de solicitações da AWS API na Referência AWS geral.

É sua responsabilidade criptografar e proteger qualquer comunicação que seus aplicativos testados façam e quaisquer aplicativos extras instalados no processo de execução de testes no dispositivo.

Criptografia em repouso

O recurso de teste do navegador de desktop do Device Farm oferece suporte à criptografia em repouso para artefatos gerados durante os testes.

Os dados de teste do dispositivo móvel físico do Device Farm não são criptografados em repouso.

Retenção de dados

Os dados no Device Farm são retidos por um tempo limitado. Depois que o período de retenção expira, os dados são removidos do armazenamento de backup do Device Farm.

Tipo de conteúdo	Período de retenção (dias)	Período de retenção de metadados (dias)
Aplicativos carregados	30	30
Pacotes de teste carregados	30	30
Logs	400	400
Gravações de vídeo e outros artefatos	400	400

É sua responsabilidade arquivar qualquer conteúdo que você queira reter por períodos mais longos.

Gerenciamento de dados

Os dados no Device Farm são gerenciados de forma diferente, dependendo de quais recursos são usados. Esta seção explica como os dados são gerenciados durante e após o uso do Device Farm.

Teste do navegador de desktop

As instâncias usadas durante as sessões do Selenium não são salvas. Todos os dados gerados como resultado de interações do navegador são descartados quando a sessão termina.

Atualmente, esse recurso oferece suporte à criptografia em repouso para artefatos gerados durante o teste.

Teste de dispositivo físico

As seções a seguir fornecem informações sobre as etapas AWS necessárias para limpar ou destruir dispositivos depois de usar o Device Farm.

Os dados de teste do dispositivo móvel físico do Device Farm não são criptografados em repouso.

Frotas de dispositivo público

Após a conclusão da execução do teste, o Device Farm realiza uma série de tarefas de limpeza em cada dispositivo da frota pública de dispositivos, incluindo a desinstalação do seu aplicativo. Se não conseguirmos verificar a desinstalação do aplicativo ou qualquer uma das outras etapas de limpeza, o dispositivo receberá uma redefinição de fábrica antes de ser recolocado em uso.

> **Note**
>
> Em alguns casos, é possível que os dados persistam entre as sessões, especialmente se você usar o sistema do dispositivo fora do contexto do seu aplicativo. Por esse motivo, e como o Device Farm captura vídeos e logs de atividades que ocorrem durante o uso de cada dispositivo, recomendamos que você não insira informações confidenciais (por exemplo, conta do Google ou ID da Apple), informações pessoais e outros detalhes sensíveis à segurança durante o teste automatizado e as sessões de acesso remoto.

Dispositivos privados

Após a expiração ou o encerramento do contrato de dispositivos privados, o dispositivo é removido do uso e destruído de maneira segura de acordo com políticas de destruição da AWS. Para obter mais informações, consulte Dispositivos privados no AWS Device Farm.

Gerenciamento de chaves

Atualmente, o Device Farm não oferece nenhum gerenciamento de chaves externas para criptografia de dados, em repouso ou em trânsito.

Privacidade do tráfego entre redes

O Device Farm pode ser configurado, apenas para dispositivos privados, para usar os endpoints da Amazon VPC para se conectar aos seus recursos na AWS. O acesso a qualquer AWS infraestrutura não pública associada à sua conta (por exemplo, EC2 instâncias da Amazon sem um endereço IP público) deve usar um endpoint Amazon VPC. Independentemente da configuração do endpoint da VPC, o Device Farm isola seu tráfego de outros usuários em toda a rede do Device Farm.

Não é garantido que suas conexões fora da AWS rede sejam seguras ou protegidas, e é sua responsabilidade proteger todas as conexões de internet que seus aplicativos fizerem.

Resiliência 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, conectadas com baixa latência, throughput elevado 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 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`, é altamente recomendável que você implemente processos de backup e recuperação. O Device Farm não deve ser a única fonte de qualquer conteúdo enviado.

O Device Farm não oferece garantias quanto à disponibilidade de dispositivos públicos. Esses dispositivos são inseridos e retirados do grupo de dispositivos públicos dependendo de diversos fatores, como a taxa de falhas e o status de quarentena. Não recomendamos que você dependa da disponibilidade de nenhum dispositivo do grupo de dispositivos públicos.

Segurança da infraestrutura em AWS Device Farm

Como serviço gerenciado, AWS Device Farm é protegido pela segurança de rede AWS global. Para obter informações sobre serviços AWS de segurança e como AWS proteger a infraestrutura, consulte AWS Cloud Security. Para projetar seu AWS ambiente usando as melhores práticas de segurança de infraestrutura, consulte Proteção de infraestrutura no Security Pillar AWS Well-Architected Framework.

Você usa chamadas de API AWS publicadas para acessar o Device Farm pela rede. Os clientes devem oferecer compatibilidade com:

- Transport Layer Security (TLS). Exigimos TLS 1.2 e recomendamos TLS 1.3.
- Conjuntos de criptografia com perfect forward secrecy (PFS) como DHE (Ephemeral Diffie-Hellman) ou ECDHE (Ephemeral Elliptic Curve Diffie-Hellman). A maioria dos sistemas modernos, como Java 7 e versões posteriores, comporta esses modos.

Além disso, as solicitações devem ser assinadas usando um ID da chave de acesso e uma chave de acesso secreta associada a uma entidade principal do IAM. Ou é possível usar o AWS Security Token Service (AWS STS) para gerar credenciais de segurança temporárias para assinar solicitações.

Segurança da infraestrutura para teste de dispositivos físicos

Os dispositivos são fisicamente separados durante o teste de dispositivos físicos. O isolamento da rede impede a comunicação entre dispositivos por meio de redes sem fios.

Os dispositivos públicos são compartilhados, e o Device Farm faz o melhor esforço para manter os dispositivos seguros ao longo do tempo. Determinadas ações, como tentativas para adquirir direitos de administrador completos em um dispositivo (uma prática referida como enraizamento ou fuga), faz com que os dispositivos públicos sejam colocados em quarentena. Eles são removidos do grupo público automaticamente e transferidos para a análise manual.

Os dispositivos privados só podem ser acessados por AWS contas explicitamente autorizadas a fazer isso. O Device Farm isola fisicamente esses dispositivos de outros dispositivos e os mantém em uma rede separada.

Em dispositivos gerenciados de forma privada, os testes podem ser configurados para usar um endpoint Amazon VPC para proteger conexões dentro e fora da sua conta. AWS

Segurança da infraestrutura para teste de navegador de desktop

Quando você usa o recurso de teste de navegador de desktop, todas as sessões de teste são separadas umas das outras. As instâncias do Selenium não podem se comunicar de forma cruzada sem um terceiro intermediário, externo a. AWS

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

Você é responsável por garantir que cada instância de teste do Device Farm tenha acesso seguro aos recursos que testa. Por padrão, as instâncias de teste do navegador de desktop do Device Farm têm acesso à Internet pública. Quando você anexa sua instância a uma VPC, ela se comporta como qualquer outra instância do EC2, com acesso aos recursos determinados pela configuração da VPC e os respectivos componentes de rede associados. A AWS fornece grupos de segurança e listas de controle de acesso (ACLs) de rede para aumentar a segurança em sua VPC. Os grupos de segurança controlam o tráfego de entrada e saída de seus recursos, e a rede ACLs controla o tráfego de entrada e saída de suas sub-redes. Os grupos de segurança fornecem controle de acesso suficiente para a maioria das sub-redes. Você pode usar a rede ACLs se quiser uma camada adicional de segurança para sua VPC. Para obter diretrizes gerais sobre as melhores práticas de segurança ao usar a Amazon VPCs, consulte as melhores práticas de segurança para sua VPC no Guia do usuário da Amazon Virtual Private Cloud.

Análise e gerenciamento de vulnerabilidades de configuração no Device Farm

O Device Farm permite que você execute software que não é ativamente mantido ou corrigido pelo fornecedor, como o fornecedor do sistema operacional, o fornecedor do hardware ou a operadora de telefonia. A Device Farm se esforça ao máximo para manter o software atualizado, mas não garante que qualquer versão específica do software em um dispositivo físico esteja atualizada, permitindo que softwares potencialmente vulneráveis sejam colocados em uso.

Por exemplo, se um teste for realizado em um dispositivo executando o Android 4.4.2, o Device Farm não garante que o dispositivo tenha sido corrigido contra a vulnerabilidade no Android conhecida como. StageFright É responsabilidade do fornecedor (e às vezes da operadora) do dispositivo fornecer atualizações de segurança para os dispositivos. Não há garantias de que um aplicativo mal-intencionado que use essa vulnerabilidade seja capturado pela nossa quarentena automatizada.

Os dispositivos privados são mantidos de acordo com seu contrato com AWS.

O Device Farm se esforça ao máximo para impedir que as aplicações do cliente realizem ações como root ou jailbreak. O Device Farm remove os dispositivos que estão em quarentena do pool público até que sejam revisados manualmente.

Você é responsável por manter atualizadas todas as bibliotecas ou versões de software que usar em seus testes, como wheels do Python e gemas do Ruby. O Device Farm recomenda que você atualize suas bibliotecas de teste.

Esses recursos podem ajudar a manter as dependências de teste atualizadas:

- Para obter informações sobre como proteger gemas Ruby, consulte Práticas de segurança no RubyGems site.
- Para obter informações sobre o pacote de segurança usado pela Pipenv e aprovado pela Python Packaging Authority para verificar seu gráfico de dependências em busca de vulnerabilidades conhecidas, consulte Detecção de vulnerabilidades de segurança em. GitHub
- Para obter informações sobre o verificador de dependência Maven do Open Web Application Security Project (OWASP), consulte OWASP no site da DependencyCheckOWASP.

É importante lembrar que, mesmo que um sistema automatizado não acredite que haja problemas de segurança conhecidos, isso não significa que não haja problemas de segurança. Sempre tenha cuidado ao usar bibliotecas ou ferramentas de terceiros e verifique as assinaturas criptográficas quando possível ou razoável.

Resposta a incidentes no Device Farm

O Device Farm monitora continuamente os dispositivos em busca de comportamentos que possam indicar problemas de segurança. Se AWS for informado de um caso em que os dados do cliente, como resultados de testes ou arquivos gravados em um dispositivo público, podem ser acessados por outro cliente, ele AWS entra em contato com os clientes afetados, de acordo com as políticas padrão de alerta e emissão de relatórios de incidentes usadas em todos AWS os serviços.

Registrar em log e monitorar no Device Farm

Esse serviço oferece suporte AWS CloudTrail, que é um serviço que registra AWS chamadas para você Conta da AWS e entrega arquivos de log em um bucket do Amazon S3. Usando as informações coletadas por CloudTrail, você pode determinar quais solicitações foram feitas com sucesso Serviços da AWS, quem fez a solicitação, quando ela foi feita e assim por diante. Para saber mais sobre

CloudTrail, inclusive como ativá-lo e encontrar seus arquivos de log, consulte o Guia AWS CloudTrail do usuário.

Para obter informações sobre como usar CloudTrail com o Device Farm, consulteRegistro em log de chamadas de API do AWS Device Farm com o AWS CloudTrail.

Práticas recomendadas de segurança para o Device Farm

O Device Farm oferece vários recursos de segurança que devem ser considerados ao desenvolver e implementar suas próprias políticas de segurança. As práticas recomendadas a seguir são diretrizes gerais e não representam uma solução completa de segurança. Como essas práticas recomendadas podem não ser adequadas ou suficientes para o seu ambiente, trate-as como considerações úteis em vez de prescrições.

- Conceda a qualquer sistema de integração contínua (CI) que você usar o mínimo de privilégios possível no IAM. Considere o uso de credenciais temporárias para cada teste de sistema de CI para que, mesmo que um sistema de CI esteja comprometido, ele não possa fazer solicitações falsas. Para obter mais informações sobre credenciais temporárias, consulte o Guia do usuário do IAM.
- Use comandos adb em um ambiente de teste personalizado para limpar qualquer conteúdo criado pelo aplicativo. Para obter mais informações sobre ambientes de teste personalizados, consulte Ambientes de teste personalizados

Limites do AWS Device Farm

Tópicos

Limites do serviço

- 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. É possível aumentar esse número mediante solicitação, que é avaliada caso a caso pela equipe de atendimento.
- Não há limite para o número de execuções que você pode programar. Observe que elas só podem permanecer na fila por até 24 horas.
- Há um limite fixo de 150 minutos de duração para uma sessão de acesso remoto.
- Há um limite fixo 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 (trabalhos) que podem executar testes em paralelo a qualquer momento é igual à simultaneidade em nível de conta. A simultaneidade padrão em nível de conta para uso medido no Device Farm é cinco.
- É possível aumentar o limite de simultaneidade medido mediante solicitação até determinado limite, dependendo do caso de uso. A simultaneidade padrão no nível da conta para uso ilimitado é igual ao número de slots nos quais você está inscrito nessa plataforma.

  Para acessar mais informações sobre os limites padrão de simultaneidade medidos ou as cotas em geral, consulte a página Cotas.
- Uma execução de automação que não usa um ambiente de teste personalizado só pode ter até 250 casos de teste individuais. Caso contrário, a execução poderá ser ignorada.

Limites de arquivo

- O tamanho máximo do arquivo de um aplicativo que você pode carregar é de 4 GB. Observe que atualmente não aceitamos arquivos no formato. aab para Android.
- O tamanho máximo do vídeo gerado automaticamente pelo Device Farm durante a execução do teste é de 1 GB. Qualquer vídeo que exceda esse tamanho terá todo o conteúdo restante truncado. Os clientes ainda podem usar sua própria solução de gravação de vídeo, se houver, e armazená-la fora do armazenamento gerenciado do Device Farm.
- O tamanho máximo do log de dispositivos gerado automaticamente pelo Device Farm (logcat no Android ou syslog no iOS) durante a execução do teste é de 1 GB. No caso de qualquer log exceder esse tamanho, todos os logs restantes serão truncados. No caso de logs maiores que 1 GB, os clientes podem salvá-los fora do armazenamento gerenciado do Device Farm.
- O tamanho máximo cumulativo dos artefatos do cliente no modo de ambiente personalizado do Device Farm é de 1 GB. Se seus artefatos excederem esse tamanho, nenhum deles ficará disponível.
- Se o tamanho cumulativo de todos os artefatos gerados durante uma execução de teste exceder 4 GB, alguns artefatos poderão ser descartados (incluindo o vídeo, os logs do dispositivo e os artefatos do cliente).

Limites de API

- O Device Farm segue um algoritmo de token-bucket para controle de utilização das taxas de chamadas de API. Por exemplo, imagine a criação de um bucket com tokens. Cada token representa uma transação, e uma chamada de API usa até um token. Os tokens são adicionados ao bucket a uma taxa fixa (por exemplo, dez tokens por segundo), e o bucket tem uma capacidade máxima (por exemplo, cem tokens). Quando uma solicitação ou pacote chega, ele deve reivindicar um token do bucket para ser processado. Se houver tokens suficientes, a solicitação será autorizada e os tokens serão removidos. Se não houver tokens suficientes, a solicitação será atrasada ou cancelada, dependendo da implementação.

  No Device Farm, é assim que o algoritmo é implementado:

  - As solicitações da API Burst correspondem ao número máximo de solicitações às quais o serviço pode responder para uma API específica em um ID de conta de cliente especificado. Em outras palavras, é a capacidade do bucket. É possível chamar a API enquanto houver tokens restantes no bucket, e cada solicitação consome um token.

- A taxa Transactions-per-second (TPS) é a taxa mínima na qual suas solicitações de API podem ser executadas. Em outras palavras, é a taxa na qual o bucket é reabastecido com tokens por segundo. Por exemplo, se uma API tiver um número de intermitência de dez, mas um TPS de um, você poderá chamá-la dez vezes instantaneamente. No entanto, o bucket só recuperaria tokens a uma taxa de um token por segundo, ocasionando o controle de capacidade para uma chamada por segundo, a menos que você parasse de chamar a API para permitir que o bucket fosse reabastecido.

Aqui estão as tarifas do Device Farm APIs:

- Para List e Get APIs, a capacidade de solicitações da API Burst é `50` e a taxa Transactions-per-second (TPS) é. `10`
- Para todas as outras APIs, a capacidade de solicitações da API Burst é`10`, e a taxa Transactions-per-second (TPS) é. `1`

Limites de endpoint do Appium

Os limites a seguir se aplicam a todas as sessões de endpoint da Appium. Para perguntas e orientações sobre a melhor forma de lidar com os limites, abra um caso de suporte.

- Cada comando do Appium tem um limite de duração de execução de 4 minutos, após o qual o comando expira.
- O endpoint aceita tamanhos de carga de entrada de até 20 MB e permite tamanhos de carga de saída de até 20 MB. Qualquer solicitação com um tamanho de entrada ou saída maior que esse receberá um WebDriver erro de`'unsupported operation'`.
- As solicitações são executadas sequencialmente no dispositivo na ordem em que são recebidas. Como resultado, é altamente recomendável enviar comandos sequencialmente e aguardar a resposta de cada comando antes de enviar um novo. Dito isso, certos comandos do servidor Appium podem ser enviados em paralelo, especificamente:
  - Obter status
  - Obtenha sessões
- O endpoint não oferece suporte ao WebDriver BiDi protocolo no momento.
- O endpoint não oferece suporte a plug-ins ou drivers Appium além dos XCUITest drivers e. UIAutomator2

- No máximo 3 aplicativos podem ser usados como aplicativos auxiliares com uma solicitação de criação de sessão de acesso remoto. Dito isso, não há limite de quantos aplicativos podem ser instalados durante uma sessão usando a `InstallToRemoteAccessSession`API.

Limites variáveis de ambiente personalizados

Os limites a seguir se aplicam a todas as variáveis de ambiente personalizadas. Para perguntas e orientações sobre a melhor forma de lidar com os limites, abra um caso de suporte.

- No máximo 32 variáveis podem ser configuradas em um determinado projeto ou execução do Device Farm.
- Os nomes das variáveis não podem exceder 256 caracteres.
- Os nomes das variáveis estão sujeitos às limitações impostas pelo`bash`. Ou seja, eles devem conter apenas caracteres alfanuméricos e sublinhados e não podem começar com um número.
- Os nomes das variáveis que começam com `$DEVICEFARM_` são reservados para uso interno do serviço.
- Os valores das variáveis não podem exceder 256 caracteres.
- As variáveis de ambiente não podem ser usadas para configurar a seleção computacional do host de teste no arquivo de especificação de teste.

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 on GitHub.

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

Tópicos

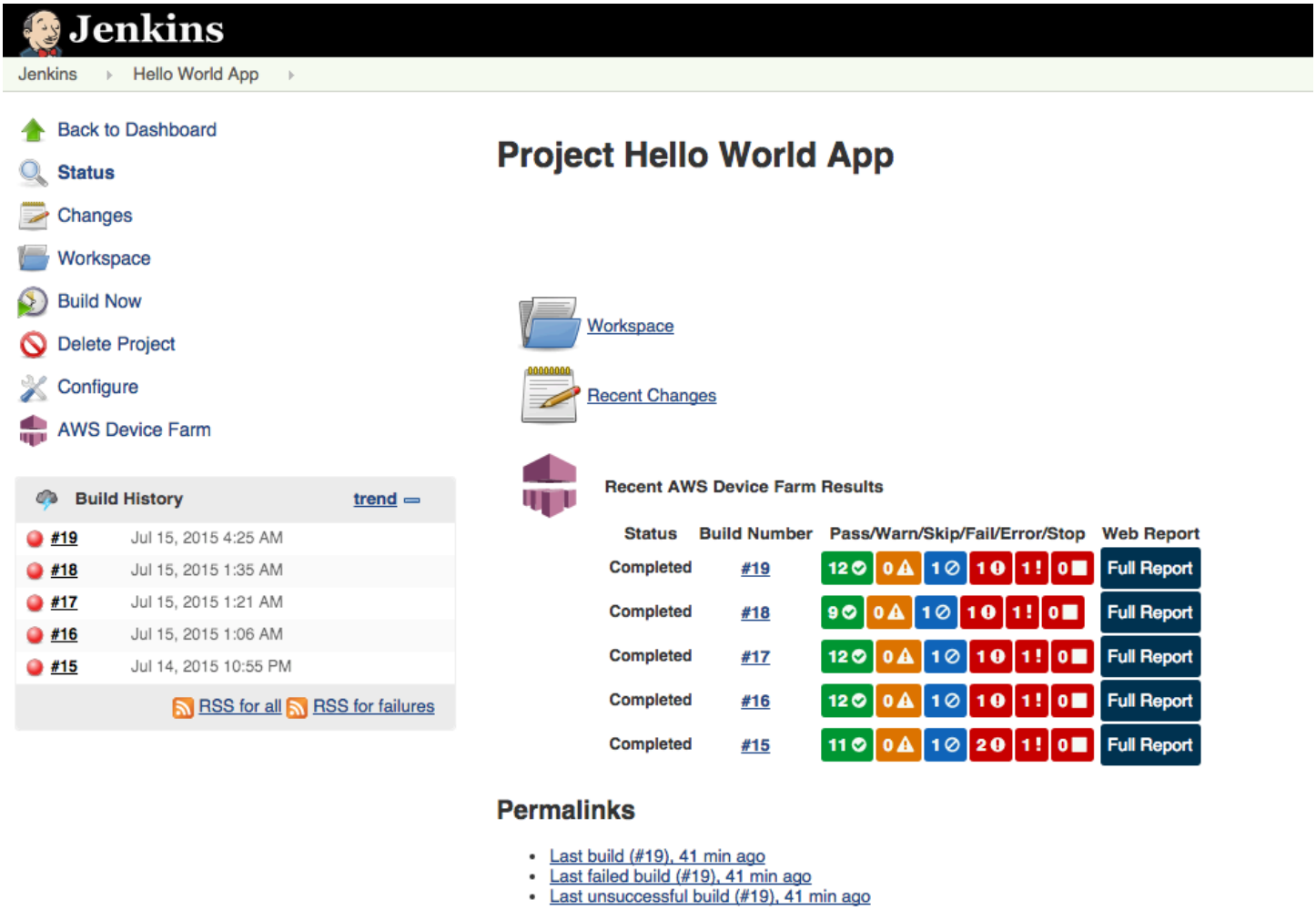
Integrar o Device Farm a um servidor Jenkins CI

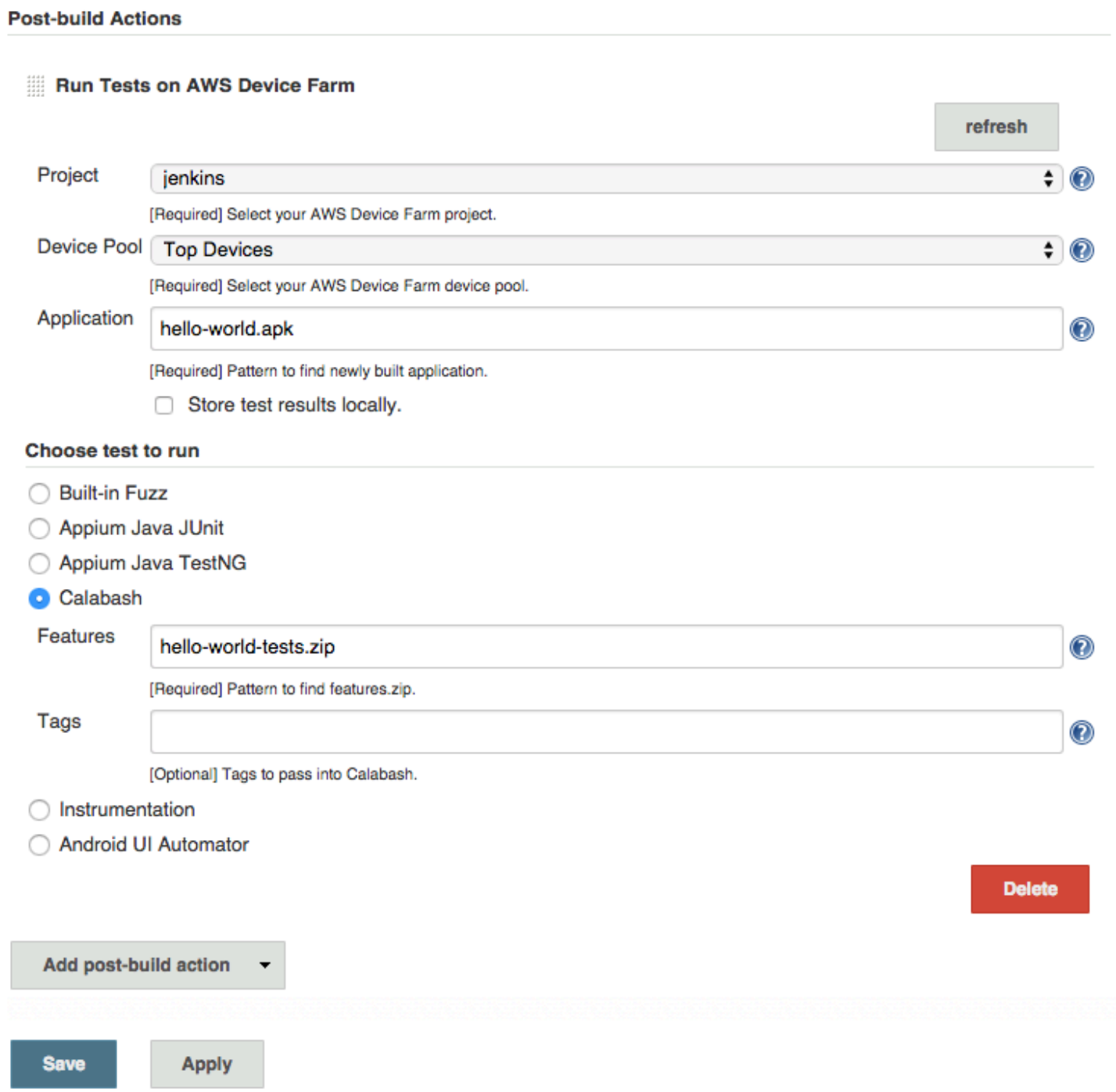
O plug-in do Jenkins CI oferece a funcionalidade do AWS Device Farm por meio do seu próprio servidor de integração contínua (CI) Jenkins. Para obter mais informações, consulte Jenkins (software).

> **Note**
>
> Para baixar o plug-in Jenkins, acesse GitHube siga as instruções emEtapa 1: Instalar o plug-in do Jenkins CI para o AWS Device Farm.

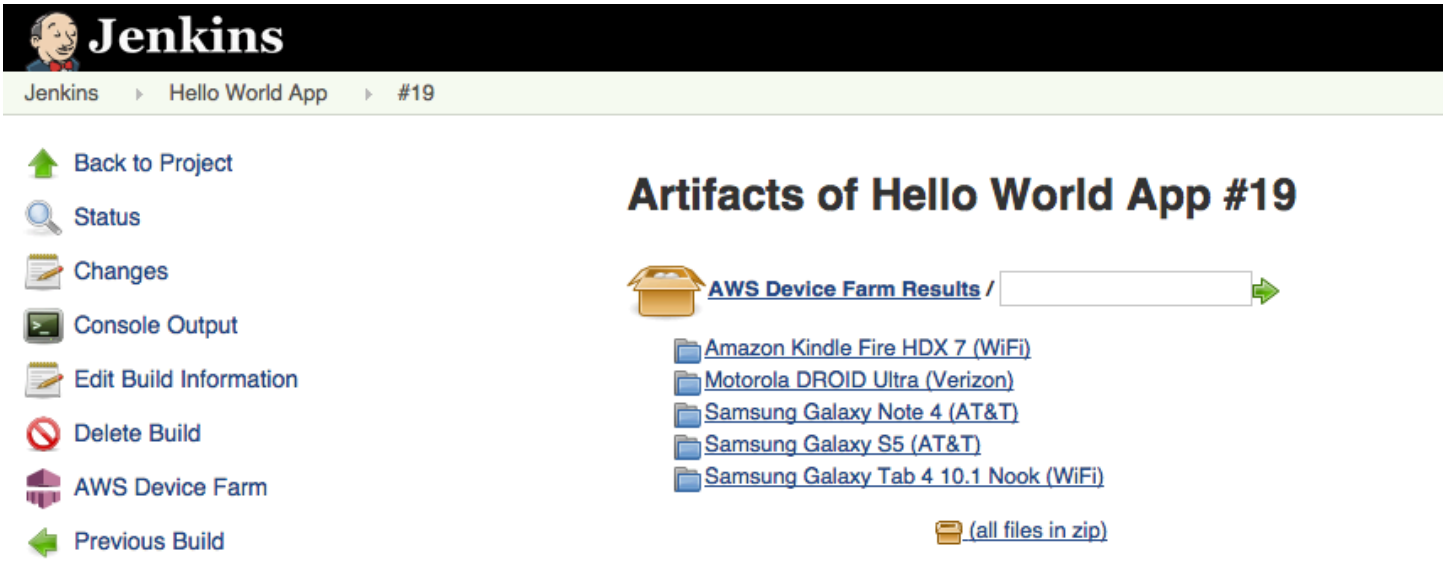
Esta seção contém uma série de procedimentos para configurar e usar o plug-in Jenkins CI com o AWS Device Farm.

As imagens a seguir mostram os recursos do plug-in Jenkins CI.


Jenkins
Jenkins ▸ Hello World App ▸
Back to Dashboard
Status
Changes
Workspace
Build Now
Delete Project
Configure
AWS Device Farm
Build History trend
#19 Jul 15, 2015 4:25 AM
#18 Jul 15, 2015 1:35 AM
#17 Jul 15, 2015 1:21 AM
#16 Jul 15, 2015 1:06 AM
#15 Jul 14, 2015 10:55 PM
RSS for all RSS for failures
Project Hello World App
Workspace
Recent Changes
Recent AWS Device Farm Results
Status Build Number Pass/Warn/Skip/Fail/Error/Stop Web Report
Completed #19 12 0 1 1 1 0 Full Report
Completed #18 9 0 1 1 1 0 Full Report
Completed #17 12 0 1 1 1 0 Full Report
Completed #16 12 0 1 1 1 0 Full Report
Completed #15 11 0 1 2 1 0 Full Report
Permalinks
Last build (#19), 41 min ago
Last failed build (#19), 41 min ago
Last unsuccessful build (#19), 41 min ago



Esse plug-in também pode abrir todos os artefatos de teste (logs, capturas de tela etc.) localmente:



Tópicos

- Dependências
- Etapa 1: Instalar o plug-in do Jenkins CI para o AWS Device Farm
- Etapa 2: Criar um AWS Identity and Access Management usuário para seu plug-in Jenkins CI para AWS Device Farm
- Etapa 3: configurar o plug-in Jenkins CI pela primeira vez no AWS Device Farm
- Etapa 4: usar o plug-in em um trabalho do Jenkins

Dependências

O plug-in Jenkins CI requer o AWS Mobile SDK 1.10.5 ou posterior. Para obter mais informações e instalar o SDK, consulte AWS Mobile SDK.

Etapa 1: Instalar o plug-in do Jenkins CI para o AWS Device Farm

Há duas opções para instalar o plug-in de integração contínua (CI) do Jenkins para o AWS Device Farm. Você pode procurar o plug-in na caixa de diálogo Available Plugins (Plug-ins disponíveis) na interface do usuário da web do Jenkins ou você pode fazer download do arquivo `hpi` e instalá-lo por meio do Jenkins.

Instalação por meio da interface de usuário do Jenkins

1. Encontre o plug-in na interface do usuário do Jenkins, escolha Manage Jenkins (Gerenciar Jenkins), Manage Plugins (Gerenciar plug-ins) e Available (Disponível).
2. Pesquise por aws-device-farm.
3. Instale o plug-in AWS Device Farm.
4. Verifique se o plug-in pertence ao usuário `Jenkins`.
5. Reinicie o Jenkins.

Faça download do plug-in.

1. Baixe o `hpi` arquivo diretamente de http://updates.jenkins-ci. org/latest/aws-device-farm.hpi.
2. Verifique se o plug-in pertence ao usuário `Jenkins`.
3. Instale o plug-in usando uma das seguintes opções:
   - Faça upload do plug-in escolhendo Manage Jenkins (Gerenciar Jenkins), Manage Plugins (Gerenciar plug-ins), Advanced (Avançado) e Upload plugin (Fazer upload do plug-in).
   - Coloque o arquivo `hpi` no diretório do plug-in Jenkins (normalmente `/var/lib/jenkins/plugins`).
4. Reinicie o Jenkins.

Etapa 2: Criar um AWS Identity and Access Management usuário para seu plug-in Jenkins CI para AWS Device Farm

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

Para criar um usuário do IAM, consulte Creating an IAM User (Console de gerenciamento da AWS). Lembre-se de gerar uma chave de acesso para cada usuário e fazer download ou salvar as credenciais de segurança do usuário. Você precisará das credenciais posteriormente.

Dê ao usuário do IAM permissão para acessar o Device Farm

Para dar permissão ao usuário do IAM para acessar o Device Farm, crie uma nova política de acesso no IAM e atribua a política de acesso ao usuário do IAM da seguinte forma.

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

Para criar a política de acesso no IAM

1. Abra o console do IAM em https://console.aws.amazon.com/iam/.
2. Selecione Políticas.
3. Escolha Create Policy. (Se aparecer um botão Get Started, selecione-o e, em seguida, Create Policy.)
4. Próximo a Create Your Own Policy, escolha Select.
5. Em Policy Name (Nome da política), digite um nome para política (por exemplo, **`AWSDeviceFarmAccessPolicy`**).
6. Em Descrição, digite uma descrição que ajude a associar esse usuário do IAM ao projeto do Jenkins.
7. Em Policy Document (Documento da política), digite a seguinte declaração:

   JSON

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

8. Escolha Create Policy.

Para atribuir a política de acesso ao usuário do IAM

1. Abra o console do IAM em https://console.aws.amazon.com/iam/.
2. Selecione Usuários.
3. Escolha o usuário do IAM ao qual você atribuirá a política de acesso.
4. Na área Permissions (Permissões), em Managed Policies (Políticas gerenciadas), escolha Attach Policy (Anexar política).
5. Selecione a política que você acabou de criar (por exemplo, AWSDeviceFarmAccessPolicy).
6. Escolha Attach Policy.

Etapa 3: configurar o plug-in Jenkins CI pela primeira vez no AWS Device Farm

A primeira vez em que você executar o servidor Jenkins, precisará configurar o sistema conforme a seguir.

> **Note**
>
> Se estiver usando slots de dispositivos, o recurso de slots para dispositivo estará desativado por padrão.

1. Faça login na interface de usuário web do Jenkins.
2. No lado esquerdo da tela, escolha Manage Jenkins (Gerenciar Jenkins).
3. Escolha Configure System (Configurar sistema).
4. Role para baixo até o cabeçalho AWS Device Farm.
5. Copie suas credenciais de segurança de Criar um usuário do IAM para o plug-in do Jenkins CI e cole o ID da chave de acesso e a chave de acesso secreta nas respectivas caixas.
6. Escolha Salvar.

Etapa 4: usar o plug-in em um trabalho do Jenkins

Assim que você tiver instalado o plug-in Jenkins, siga estas instruções para usar o plug-in em um trabalho do Jenkins.

1. Faça login na interface de usuário web do Jenkins.
2. Clique no trabalho que você deseja editar.
3. No lado esquerdo da tela, escolha Configurar.
4. Role para baixo até o cabeçalho Ações pós-compilação.
5. Clique em Adicionar ação pós-compilação e selecione Executar testes no AWS Device Farm.
6. Selecione o projeto que você deseja usar.
7. Selecione o grupo de dispositivos que você deseja usar.
8. Selecione se você gostaria de ter os artefatos de teste (como logs e capturas de tela) arquivados localmente.
9. Em Aplicativo, preencha o caminho do aplicativo compilado.
10. Selecione o teste que deseja executar e preencha todos os campos obrigatórios.
11. Escolha Salvar.

Integrar o Device Farm a um sistema de compilação Gradle

O plug-in do Gradle do Device Farm oferece integração do AWS Device Farm ao sistema de compilação Gradle no Android Studio. Para obter mais informações, consulte Gradle.

> **Note**
>
> Para baixar o plug-in do Gradle, acesse GitHube siga as instruções emCriação do plug-in Gradle do Device Farm.

O plug-in Gradle do Device Farm fornece a funcionalidade do Device Farm em seu ambiente do Android Studio. Você pode iniciar testes em telefones e tablets Android reais hospedados pelo Device Farm.

Esta seção contém uma série de procedimentos para configurar e usar o plug-in Gradle do Device Farm.

Tópicos

- Dependências
- Etapa 1: Criação do plug-in Gradle do AWS Device Farm
- Etapa 2: Configurar o plug-in Gradle do AWS Device Farm

Dependências

Runtime

- O plug-in Device Farm Gradle requer o AWS Mobile SDK 1.10.15 ou posterior. Para obter mais informações e instalar o SDK, consulte AWS Mobile SDK.
- Android tools builder test api 0.5.2
- Apache Commons Lang3 3.3.4

For Unit Tests (Para testes de unidade)

- Testng 6.8.8
- Jmockit 1.19
- Android gradle tools 1.3.0

Etapa 1: Criação do plug-in Gradle do AWS Device Farm

Esse plug-in fornece integração do AWS Device Farm com o sistema de compilação Gradle no Android Studio. Para obter mais informações, consulte Gradle.

> **Note**
>
> A criação desse plug-in é opcional. O plug-in é publicado por meio do Maven Central. Se deseja permitir que o Gradle faça download do plug-in diretamente, ignore esta etapa e vá para Etapa 2: Configurar o plug-in Gradle do AWS Device Farm.

Para criar o plug-in

1. Acesse GitHube clone o repositório.
2. Crie o plug-in usando `gradle install`.

   O plug-in é instalado no seu repositório maven local.

Próxima etapa: Etapa 2: Configurar o plug-in Gradle do AWS Device Farm

Etapa 2: Configurar o plug-in Gradle do AWS Device Farm

Se você ainda não tiver feito isso, clone o repositório e instale o plug-in usando o procedimento descrito em: Criação do plug-in Gradle do Device Farm.

Para configurar o plug-in Gradle do AWS Device Farm

1. Adicione o artefato do plug-in à sua lista de dependências em `build.gradle`.

```
    buildscript {

        repositories {
            mavenLocal()
            mavenCentral()
        }

        dependencies {
            classpath 'com.android.tools.build:gradle:1.3.0'
            classpath 'com.amazonaws:aws-devicefarm-gradle-plugin:1.0'
        }
    }
```

2. Configure o plug-in em seu arquivo `build.gradle`. A configuração específica de teste a seguir deve servir de guia:

```
apply plugin: 'devicefarm'

devicefarm {

    // Required. The project must already exist. You can create a project in the
 AWS Device Farm console.
    projectName "My Project" // required: Must already exist.

    // Optional. Defaults to "Top Devices"
    // devicePool "My Device Pool Name"

    // Optional. Default is 150 minutes
    // executionTimeoutMinutes 150

    // Optional. Set to "off" if you want to disable device video recording during
 a run. Default is "on"
```

```
   // videoRecording "on"

   // Optional. Set to "off" if you want to disable device performance monitoring
during a run. Default is "on"
   // performanceMonitoring "on"

   // Optional. Add this if you have a subscription and want to use your unmetered
slots
   // useUnmeteredDevices()

   // Required. You must specify either accessKey and secretKey OR roleArn.
roleArn takes precedence.
   authentication {
       accessKey "AKIAIOSFODNN7EXAMPLE"
       secretKey "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"

       // OR

       roleArn "arn:aws:iam::111122223333:role/DeviceFarmRole"
   }

   // Optionally, you can
   // - enable or disable Wi-Fi, Bluetooth, GPS, NFC radios
   // - set the GPS coordinates
   // - specify files and applications that must be on the device when your test
runs
   devicestate {
       // Extra files to include on the device.
       // extraDataZipFile file("path/to/zip")

       // Other applications that must be installed in addition to yours.
       // auxiliaryApps files(file("path/to/app"), file("path/to/app2"))

       // By default, Wi-Fi, Bluetooth, GPS, and NFC are turned on.
       // wifi "off"
       // bluetooth "off"
       // gps "off"
       // nfc "off"

       // You can specify GPS location. By default, this location is 47.6204,
-122.3491
       // latitude 44.97005
       // longitude -93.28872
   }
```

```
    // By default, the Instrumentation test is used.
    // If you want to use a different test type, configure it here.
    // You can set only one test type (for example, Calabash, Fuzz, and so on)

    // Fuzz
    // fuzz { }

    // Calabash
    // calabash { tests file("path-to-features.zip") }

}
```

3. Execute o teste do Device Farm usando a seguinte tarefa: `gradle devicefarmUpload`.

   A saída da compilação imprimirá um link para o console do Device Farm, onde você poderá monitorar a execução do teste.

Próxima etapa: Gerar um usuário do IAM no plug-in do Gradle do Device Farm

Etapa 3: gerar um usuário do IAM no plug-in do Gradle do Device Farm

AWS Identity and Access Management (IAM) ajuda você a gerenciar permissões e políticas para trabalhar com AWS recursos. Este tópico orienta você na geração de um usuário do IAM com permissões para acessar os recursos do AWS Device Farm.

Se ainda não tiver feito isso, conclua as etapas 1 e 2 antes de gerar um usuário do IAM.

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

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

Para criar um novo usuário com a política de acesso adequada no IAM

1. Abra o console do IAM em https://console.aws.amazon.com/iam/.
2. Selecione Usuários.
3. Escolha Create New Users (Criar novos usuários).
4. Digite o nome de usuário de sua escolha.

   Por exemplo, .**`GradleUser`**
5. Escolha Criar.
6. Escolha Download Credentials (Fazer download de credenciais) e as salve em um local onde você possa recuperá-las facilmente depois.
7. Escolha Fechar.
8. Escolha o nome de usuário na lista.
9. Em Permissions (Permissões), expanda o cabeçalho Inline Policies (Políticas em linha) clicando na seta para baixo à direita.
10. Escolha Clique aqui onde está escrito: Não há políticas em linha para mostrar. Para criar uma, clique aqui.
11. Na tela Definir permissões, escolha Política personalizada.
12. Escolha Selecionar.
13. Dê um nome à política, como **`AWSDeviceFarmGradlePolicy`**.
14. Cole a política a seguir em Documento da política.

    JSON

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

15. Escolha Aplicar política.

Próxima etapa: Configuração de tipos de teste.

Para obter mais informações, consulte Creating an IAM User (Console de gerenciamento da AWS) ou Configurar.

Etapa 4: Configuração dos tipos de teste

Por padrão, o plug-in do AWS Device Farm Gradle executa o teste Instrumentação para Android e AWS Device Farm. Se deseja executar seus próprios testes ou especificar outros parâmetros, você pode optar por configurar um tipo de teste. Este tópico fornece informações sobre cada tipo de teste disponível e o que você precisa fazer no Android Studio a fim de configurá-lo para uso. Para obter mais informações sobre os tipos de teste disponíveis no Device Farm, consulte Frameworks de teste e testes integrados no AWS Device Farm.

Se ainda não tiver feito isso, execute as etapas de 1 a 3 antes de configurar os tipos de teste.

> **Note**
>
> Se estiver usando slots de dispositivos, o recurso de slots para dispositivo estará desativado por padrão.

Appium

O Device Farm fornece suporte para Appium Java e JUnit TestNG para Android.

- Appium (em Java ()) JUnit
- Appium (em Java [TestNG])

Você pode escolher `useTestNG()` ou `useJUnit()`. `JUnit` é o padrão e não precisa ser especificado explicitamente.

```
appium {
    tests file("path to zip file") // required
    useTestNG() // or useJUnit()
}
```

Integrado: Fuzz

O Device Farm fornece um tipo de teste de fuzz incorporado, que envia aleatoriamente eventos da interface do usuário para dispositivos e, em seguida, relata os resultados.

```
    fuzz {

       eventThrottle 50 // optional default
       eventCount 6000  // optional default
       randomizerSeed 1234 // optional default blank

     }
```

Para obter mais informações, consulte Executar o teste fuzz incorporado do Device Farm (Android e iOS).

Instrumentação

O Device Farm fornece suporte para instrumentação (EspressoJUnit, Robotium ou qualquer teste baseado em instrumentação) para Android. Para obter mais informações, consulte Instrumentação para Android e AWS Device Farm.

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

```
    instrumentation {

        filter "test filter per developer docs" // optional

    }
```

Histórico de documentação do AWS Device Farm

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

Alteração	Descrição	Alterado em
Suporte para endpoints Appium	O Device Farm agora oferece um endpoint Appium totalmente gerenciado para testes remotos de dispositivos, permitindo desenvolvimento e depuração rápidos de testes. Isso complementa o método de execução existente no lado do servidor, em que os testes são carregados e executados diretamente no Device Farm. Embora a execução no lado do servidor seja ideal para CI/CD pipelines e testes em grande escala, o novo endpoint local Appium permite iteração e desenvolvimento mais rápidos de testes em dispositivos reais.	17 de novembro de 2025
Melhorias no host de teste do iOS	O Device Farm agora oferece suporte a uma experiência atualizada para o ambiente de teste do iOS, permitindo a consistência nas configurações entre os testes do Android e do iOS. Para saber mais, consulte Hosts para ambientes de teste personalizados. Além disso, as informações relacionadas aos hosts de teste do Android desativados foram removidas. Os usuários do Android são incentivados a usar os hosts de teste do Amazon Linux 2.	31 de outubro de 2025
AL2 apoio	O Device Farm agora é compatível com o ambiente de AL2 teste para Android. Saiba mais sobre AL2.	6 de novembro de 2023
Migração de ambientes de teste padrão para personalizados	Guia de migração atualizado para documentar a descontinuação dos testes do modo padrão em dezembro de 2023.	3 de setembro de 2023

Alteração	Descrição	Alterado em
Suporte a ENI de VPC	O Device Farm agora permite que dispositivos privados usem o recurso de conectividade ENI de VPC para ajudar os clientes a se conectarem com segurança a seus endpoints privados hospedados na AWS, no software on-premise ou em outro provedor de nuvem. Saiba mais sobre ENI de VPC.	15 de maio de 2023
Atualizações da interface do usuário do Polaris	O console do Device Farm agora é compatível com a estrutura Polaris.	28 de julho de 2021
Suporte ao Python 3	O Device Farm agora é compatível com Python 3 em testes de modo personalizado. Saiba mais sobre como usar o Python 3 nos pacotes de teste: • Appium (Python) • Appium (Python)	20 de abril de 2020
Novas informaçõ es de segurança e informações sobre AWS recursos de marcação.	Para tornar AWS os serviços de proteção mais fáceis e abrangentes, uma nova seção sobre segurança foi criada. Para ler mais, consulte Segurança em AWS Device Farm Foi adicionada uma nova seção sobre marcação no Device Farm. Para saber mais sobre atribuição de tags, consulte Marcação no Device Farm.	27 de março de 2020
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

Alteração	Descrição	Alterado em
Novo requisito para execuções de teste com XCTest	Para execuções de teste que usam a XCTest estrutura, o Device Farm agora exige um pacote de aplicativos criado para testes. Saiba mais sobre the section called “XCTest”.	4 de fevereiro de 2019
Compatibilidade com os tipos de teste do Appium Node.js e do Appium Ruby em ambientes personalizados	Agora você pode executar seus testes nos ambientes de teste personalizados do Appium Node.js e do Appium Ruby. Saiba mais sobre Frameworks de teste e testes integrados no AWS Device Farm.	10 de janeiro de 2019
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 Frameworks de teste e testes integrados no AWS Device Farm.	2 de outubro de 2018
Ambientes de teste personalizados	Com um ambiente de teste personalizado, você pode garantir que seus testes sejam executados da mesma forma que em seu ambiente local. O Device Farm agora fornece suporte para log ao vivo e streaming de vídeo, para que você possa obter feedback instantâneo sobre seus testes que são executados em um ambiente de teste personalizado. Saiba mais sobre Ambiente de teste personalizado no AWS Device Farm..	16 de agosto de 2018

Alteração	Descrição	Alterado em
Support para usar o Device Farm como provedor AWS CodePipeline de testes	Agora você pode configurar um pipeline AWS CodePipeline para usar as execuções do AWS Device Farm como ações de teste em seu processo de lançamento. CodePipeline permite que você vincule rapidamente seu repositório às etapas de criação e teste para obter um sistema de integração contínua personalizado de acordo com suas necessidades. Saiba mais sobre Integração do AWS Device Farm em um estágio de CodePipeline teste.	19 de julho de 2018
Suporte para dispositivos privados	Agora você pode usar dispositivos privados para programar execuções de teste e iniciar sessões de acesso remoto. Você pode gerenciar perfis e configurações para esses dispositivos, criar endpoints da Amazon VPC para testar aplicações privadas e criar sessões de depuração remota. Saiba mais sobre Dispositivos privados no AWS Device Farm.	2 de maio de 2018
Compatibilidade com o Appium 1.6.3	Agora você pode definir uma versão do Appium para testes personalizados do Appium.	21 de março de 2017
Definição do tempo limite de execução de testes	Você pode definir o tempo limite para a execução de um teste ou de todos os testes em um projeto. Saiba mais sobre Definir o tempo limite 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 Simular 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

Alteração	Descrição	Alterado em
Sessões de acesso remoto	Agora você pode acessar e interagir remotamente com um único dispositivo no console. Saiba mais sobre Acesso remoto.	19 de abril de 2016
Autoatendimento para slots de dispositivo	Agora você pode comprar slots de dispositivos usando a Console de gerenciamento da AWS AWS Command Line Interface, a ou a API. Saiba mais sobre como Comprar um slot de dispositivo no Device Farm.	22 de março de 2016
Como interromper execuções de teste	Agora você pode interromper as execuções de teste usando a Console de gerenciamento da AWS AWS Command Line Interface, a ou a API. Saiba mais sobre como Interromper uma execução no AWS Device Farm.	22 de março de 2016
Novos tipos de teste de XCTest interface do usuário	Agora você pode executar testes personalizados de XCTest interface de usuário em aplicativos iOS. Saiba mais sobre o tipo de teste Integrando a XCTest interface do usuário para iOS com o Device Farm.	8 de março de 2016
Novos tipos de teste do Appium Python	Você já pode executar testes personalizados do Appium Python no Android, no iOS e em aplicativos web. Saiba mais sobre Frameworks de teste e testes integrados no AWS Device Farm.	19 de janeiro de 2016
Tipos de teste de aplicativos web	Agora você pode executar testes personalizados Appium Java e JUnit TestNG em aplicativos web. Saiba mais sobre Testes de aplicações web no AWS Device Farm.	19 de novembro de 2015
Plug-in Gradle do AWS Device Farm	Saiba mais sobre como instalar e usar o Plug-in Gradle do Device Farm.	28 de setembro de 2015
Novo teste integrado para Android: Explorer	O teste integrado Explorer percorre o aplicativo analisando cada tela como se fosse um usuário final e faz capturas de tela à medida que o examina.	16 de setembro de 2015

Alteração	Descrição	Alterado em
Nova opção compatibilidade com iOS	Saiba mais sobre como testar dispositivos iOS e executar testes iOS (inclusive XCTest) emFrameworks de teste e testes integrados no AWS Device Farm.	4 de agosto de 2015
Lançamento público inicial	Essa é a versão pública inicial do Guia do desenvolvedor do AWS Device Farm.	13 de julho de 2015

Glossário da 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.