Guía para desarrolladores

AWS Device Farm

Versión de API 2015-06-23

AWS Device Farm: Guía para desarrolladores

Table of Contents

¿Qué es AWS Device Farm?

Device Farm es un servicio de pruebas de aplicaciones que puede usar para probar e interactuar con sus aplicaciones Android, iOS y web en teléfonos y tablets físicos reales con host en Amazon Web Services (AWS).

Existen dos formas principales de utilizar Device Farm:

- Pruebas automatizadas de aplicaciones con una gran variedad de marcos de pruebas.
- Acceso remoto de dispositivos en los que puede cargar, ejecutar e interactuar con aplicaciones en tiempo real.

> **Note**
>
> Device Farm solo está disponible en la región `us-west-2` (Oregón).

Pruebas de aplicaciones automatizadas

Device Farm le permite cargar sus propias pruebas o utilizar pruebas de compatibilidad integradas sin scripts. Dado que las pruebas se realizan de forma automática en paralelo, en pocos minutos comienzan pruebas en varios dispositivos.

A medida que se completan las pruebas, se actualiza un informe de prueba que contiene resultados generales, registros detallados, capturas de pantalla píxel a píxel y datos de desempeño.

Device Farm admite realizar pruebas de aplicaciones nativas e híbridas para Android e iOS, incluidas las aplicaciones creadas con PhoneGap, Titanium, Xamarin, Unity y otros marcos. Admite acceso remoto de aplicaciones Android e iOS para realizar pruebas interactivas. Para obtener más información acerca de los tipos de pruebas admitidos, consulte Trabajar con tipos de pruebas en AWS Device Farm.

Interacción de acceso remoto

El acceso remoto le permite deslizar el dedo por la pantalla, realizar gestos e interactuar con un dispositivo a través del navegador web en tiempo real. Existen una serie de situaciones en las

que resulta útil la interacción en tiempo real con un dispositivo. Por ejemplo, los representantes del servicio de atención al cliente pueden guiar a los clientes sobre cómo utilizar o configurar su dispositivo. También pueden orientar a los clientes sobre cómo utilizar aplicaciones que se ejecutan en un dispositivo concreto. Puede instalar aplicaciones en un dispositivo que se ejecuta en una sesión de acceso remoto y, a continuación, reproducir los problemas del cliente o los errores notificados.

Durante una sesión de acceso remoto, Device Farm recopila detalles acerca de las acciones que tienen lugar mientras interactúa con el dispositivo. Al final de la sesión, se generan registros con estos detalles y una captura de vídeo de la sesión.

Terminología

Device Farm introduce los siguientes términos que definen la forma en que se organiza la información:

grupo de dispositivos

: Colección de dispositivos que suelen compartir características similares, tales como la plataforma, el fabricante o el modelo.

tarea

: Una solicitud a Device Farm para que pruebe una única aplicación en un único dispositivo. Una tarea contiene uno o varios conjuntos.

medición

: Se refiere a la facturación para dispositivos. Es posible que aparezcan referencias a dispositivos con o sin medidor en la documentación y en la referencia de la API. Para obtener más información, consulte los precios de AWS Device Farm.

project

: Un espacio de trabajo lógico que contiene ejecuciones, una ejecución para cada prueba de una única aplicación en uno o varios dispositivos. Puede usar los proyectos para organizar los espacios de trabajo de la forma que usted elija. Por ejemplo, puede tener un proyecto organizado según el título de aplicación u otro, según la plataforma. Puede crear todos los proyectos que necesite.

report

Contiene información sobre una ejecución, que es una solicitud a Device Farm para que pruebe una única aplicación en uno o varios dispositivos. Para obtener más información, consulte Informes en AWS Device Farm.

run

Una compilación específica de la aplicación, con un conjunto específico de pruebas, que se ejecutará en un conjunto específico de dispositivos. Una ejecución produce un informe de los resultados. Una ejecución contiene una o varias tareas. Para obtener más información, consulte Ejecuciones.

session

Interacción en tiempo real con un dispositivo físico real a través del navegador web. Para obtener más información, consulte Sesiones.

conjunto

La organización jerárquica de las pruebas en un paquete de pruebas. Un conjunto contiene una o más pruebas.

test

Caso de prueba individual dentro de un paquete de pruebas.

Para obtener más información sobre Device Farm, consulte Conceptos.

Configuración

Para usar Device Farm, consulte Configuración.

Configuración de AWS Device Farm

Antes de usar Device Farm por primera vez, debe completar las tareas siguientes:

Temas

Paso 1: Inscríbase en AWS

Registro en Amazon Web Services (AWS)

Si no tiene uno Cuenta de AWS, complete los siguientes pasos para crearlo.

Para suscribirte a una Cuenta de AWS

1. Abra https://portal.aws.amazon.com/billing/signup.
2. Siga las instrucciones que se le indiquen.

   Parte del procedimiento de registro consiste en recibir una llamada telefónica e indicar un código de verificación en el teclado del teléfono.

   Cuando te registras en un Cuenta de AWS, Usuario raíz de la cuenta de AWSse crea un. El usuario raíz tendrá acceso a todos los Servicios de AWS y recursos de esa cuenta. Como práctica recomendada de seguridad, asigne acceso administrativo a un usuario administrativo y utilice únicamente el usuario raíz para ejecutar tareas que requieren acceso de usuario raíz.

Paso 2: Crea o usa un usuario de IAM en tu cuenta AWS

Le recomendamos que no utilice su cuenta AWS root para acceder a Device Farm. En su lugar, cree un usuario AWS Identity and Access Management (de IAM) (o utilice uno existente) en su AWS cuenta y, a continuación, acceda a Device Farm con ese usuario de IAM.

Para obtener más información, consulte Creación de usuarios de IAM (AWS Management Console).

Paso 3: Dar al usuario de IAM permiso para obtener acceso a Device Farm

Dé al usuario de IAM permiso para obtener acceso a Device Farm. Para ello, cree una nueva política de acceso en IAM y, a continuación, asigne la política de acceso al usuario de IAM, como se indica a continuación.

> **Note**
>
> La cuenta AWS raíz o el usuario de IAM que utilice para completar los siguientes pasos debe tener permiso para crear la siguiente política de IAM y asociarla al usuario de IAM. Para obtener más información, consulte Administración de políticas de IAM.

1. Cree una política con el siguiente cuerpo JSON. Asígnele un título descriptivo, como. *DeviceFarmAdmin*

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

   Para obtener más información acerca de cómo crear políticas de IAM, consulte Crear políticas de IAM en la guía del usuario de IAM.

2. Asocie la política de IAM que ha creado al nuevo usuario. Para obtener más información sobre cómo asociar políticas de IAM a los usuarios, consulte Agregar y quitar políticas de IAM en la guía del usuario de IAM.

Al asociar la política, se proporciona al usuario de IAM acceso a todas las acciones y recursos de Device Farm que se han asociado a ese usuario de IAM. Para obtener información acerca de cómo restringir a los usuarios de IAM a un conjunto limitado de acciones y recursos de IAM, consulte Administración de identidades y accesos en AWS Device Farm.

Siguiente paso

Ahora está listo para empezar a utilizar Device Farm. Consulte Introducción a Device Farm.

Introducción a Device Farm

Este tutorial le muestra cómo utilizar Device Farm para probar una aplicación nativa Android o iOS. Se utiliza la consola de Device Farm para crear un proyecto, cargar un archivo .apk o .ipa, ejecutar un conjunto de pruebas estándar y, a continuación, ver los resultados.

> **Note**
>
> Device Farm solo está disponible en la región `us-west-2` (Oregón).AWS

Temas

Requisitos previos

Antes de comenzar, asegúrese de que cumple los siguientes requisitos:

- Realice los pasos que se indican en Configuración. Necesita una cuenta de AWS y un usuario de AWS Identity and Access Management (IAM) con permiso de acceso a Device Farm.

- Para Android, necesita un archivo .apk (paquete de aplicaciones Android). Para iOS, necesita un archivo .ipa (archivo de aplicaciones iOS). El archivo se carga en Device Farm más adelante en este tutorial.

  > **Note**
  >
  > Asegúrese de que el archivo .ipa se ha compilado para un dispositivo iOS y no para un simulador.

- (Opcional) Necesita una prueba de uno de los marcos de pruebas compatibles con Device Farm. Deberá cargar este paquete de pruebas en Device Farm y, a continuación, ejecutar la prueba más adelante en este tutorial. (Si no dispone de un paquete de pruebas disponible, puede especificar y ejecutar un conjunto de pruebas integrado estándar). Para obtener más información, consulte Trabajar con tipos de pruebas en AWS Device Farm.

Paso 1: Iniciar sesión en la consola de

Puede utilizar la consola de Device Farm para crear y administrar proyectos y ejecuciones de las pruebas. Obtendrá información acerca de proyectos y ejecuciones más adelante en este tutorial.

- Inicie sesión en la consola de Device Farm en https://console.aws.amazon.com/devicefarm.

Paso 2: Crear un proyecto

Para probar una aplicación en Device Farm, primero debe crear un proyecto.

1. En el panel de navegación, seleccione Pruebas de dispositivos móviles y, a continuación, seleccione Proyectos.
2. En Proyectos de pruebas de dispositivos móviles, seleccione Nuevo proyecto.
3. En Crear proyecto, introduzca un Nombre del proyecto (por ejemplo, **`MyDemoProject`**).
4. Seleccione Crear.

   La consola abre la página Pruebas automatizadas del proyecto recién creado.

Paso 3: Crear y comenzar una ejecución

Ahora que ya tiene un proyecto, puede crear y, a continuación, comenzar una ejecución. Para obtener más información, consulte Ejecuciones.

1. En la página Pruebas automatizadas, seleccione Crear una nueva ejecución.
2. En la página Elegir aplicación, en Aplicación móvil, seleccione Elegir archivo y, a continuación, seleccione un archivo Android (.apk) o iOS (.ipa) de su ordenador. O bien, arrastre el archivo desde el ordenador y suéltelo en la consola.
3. Escriba un Nombre de la ejecución, como **`my first test`**. De forma predeterminada, la consola de Device Farm usa el nombre del archivo.

4. Seleccione Siguiente.
5. En la página Configurar, en Configurar marco de pruebas, seleccione uno de los marcos de prueba o conjuntos de pruebas integrados. Para obtener más información acerca de cada opción, consulte Tipos y marcos de prueba.
   - Si aún no ha empaquetado sus pruebas para Device Farm, seleccione Built-in: Fuzz para ejecutar un conjunto de pruebas estándar e integrado. Puede mantener los valores predeterminados para Recuento de eventos, Acelerador de eventos y Semilla aleatorizadora. Para obtener más información, consulte the section called “Integrado: fuzzing (Android e iOS)”.
   - Si tiene un paquete de pruebas de uno de los marcos de pruebas compatibles, seleccione el marco de prueba correspondiente y, a continuación, cargue el archivo que contiene las pruebas.
6. Seleccione Siguiente.
7. En la página Seleccionar dispositivos, en Grupo de dispositivos, seleccione Dispositivos principales.
8. Seleccione Siguiente.
9. En la página Especificar el estado del dispositivo, realice cualquiera de las opciones siguientes:
   - Para proporcionar datos adicionales para que Device Farm los utilice durante la ejecución, en Agregar datos adicionales, cargue un archivo .zip.
   - Para instalar otras aplicaciones para la ejecución, en Instalar otras aplicaciones, carga los archivos .apk o .ipa de las aplicaciones. Para cambiar el orden de instalación, arrastre y suelte los archivos.
   - Para activar las radios wifi, Bluetooth, GPS o NFC mientras se realiza la ejecución, en Definir estados de radio, seleccione las casillas correspondientes.

     > **Note**
     >
     > La configuración del estado de radio del dispositivo solo está disponible en pruebas nativas de Android en este momento.

   - Para probar el comportamiento específico de una ubicación durante la carrera, en Ubicación del dispositivo, especifique las coordenadas de Latitud y Longitud predefinidas.
   - Para preestablecer el idioma y la región del dispositivo para la ejecución, en Configuración regional del dispositivo, seleccione una configuración regional.

- Para preestablecer el perfil de red para la ejecución, en Perfil de red, seleccione un perfil seleccionado. O bien, seleccione Crear perfil de red para crear el suyo propio.

10. Seleccione Siguiente.
11. En la página Revisar e iniciar ejecución, seleccione Confirmar e iniciar ejecución.

Device Farm comenzará la ejecución tan pronto como los dispositivos estén disponibles, normalmente en unos minutos. Para ver el estado de la ejecución, en la página Pruebas automatizadas de su proyecto, seleccione el nombre de la ejecución. En la página de ejecución, en Dispositivos, cada dispositivo comienza con el icono de pendiente

en la tabla de dispositivos y, después, cambia al icono de ejecución

cuando comienza la prueba. Al finalizar cada prueba, la consola muestra un icono con el resultado de la prueba junto al nombre del dispositivo. Cuando se hayan completado todas las pruebas, el icono de pendiente situado junto a la ejecución pasará a ser el icono del resultado de la prueba.

Paso 4: Ver los resultados de la ejecución

Para ver los resultados de las pruebas de la ejecución, en la página Pruebas automatizadas de su proyecto, seleccione el nombre de la ejecución. Se mostrará una página de resumen:

- El número total de pruebas, por resultado.
- Lista de las pruebas con advertencias y errores únicos.
- Una lista de dispositivos y los resultados de las pruebas para cada uno de ellos.
- Todas las capturas de pantalla tomadas durante la ejecución, agrupadas por dispositivo.
- Una sección para descargar el resultado del análisis.

Para obtener más información, consulte Trabajar con informes de pruebas en Device Farm.

Pasos siguientes

Para obtener más información sobre Device Farm, consulte Conceptos.

Adquisición de una ranura para dispositivos en Device Farm

Para adquirir una ranura de dispositivos, puede utilizar la consola de Device Farm, AWS Command Line Interface (AWS CLI) o la API de Device Farm.

Temas

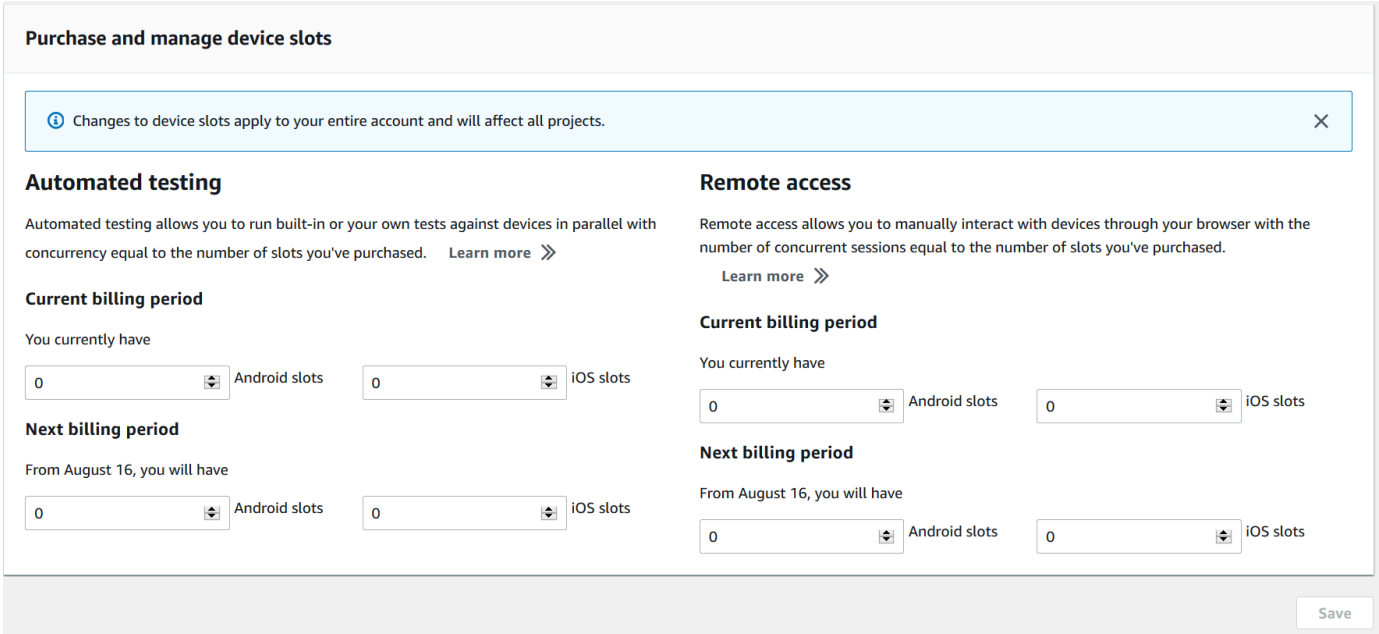
Adquisición de ranuras de dispositivos (consola)

1. Inicie sesión en la consola de Device Farm en https://console.aws.amazon.com/devicefarm.
2. En el panel de navegación, seleccione Pruebas de dispositivos móviles y, a continuación, Ranuras de dispositivos.
3. En la página Comprar y administrar ranuras de dispositivos, puede crear su propio paquete personalizado eligiendo la cantidad de ranuras para dispositivos de pruebas automatizadas y acceso remoto que desee comprar. Especifique el número de ranuras tanto para el período de facturación actual como para el siguiente.

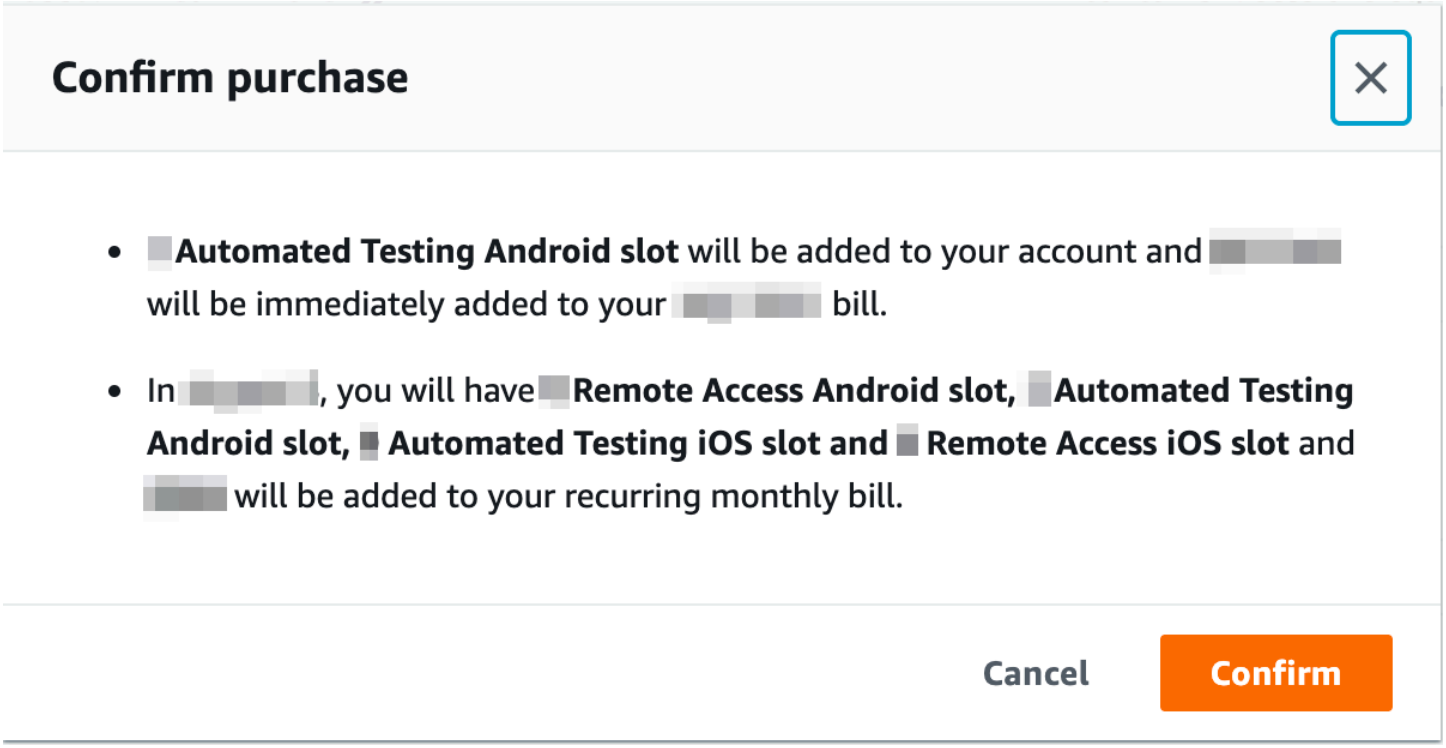
   A medida que cambie el importe de las ranuras, el texto se actualiza dinámicamente con el importe de facturación. Para obtener más información, consulte los precios de AWS Device Farm.

   > **Important**
   >
   > Si cambia el número de ranuras para dispositivos pero ve un mensaje de Contacte con nosotros o Contacte con nosotros para comprar, significa que su cuenta de AWS aún no está aprobada para comprar el número solicitado de ranuras para dispositivos.
   > Estas opciones le piden que envíe un correo electrónico al equipo de soporte de Device Farm. En el correo electrónico, especifique el número de cada tipo de dispositivo que desee comprar y para qué ciclo de facturación.

> **Note**
>
> Los cambios en las ranuras de los dispositivos a toda su cuenta y afectan a todos los proyectos.



4. Seleccione Comprar. Aparece la ventana Confirmar compra. Revise la información y, a continuación, seleccione Confirmar para completar la transacción.



En la página Comprar y administrar ranuras de dispositivos, puede ver la cantidad de ranuras para dispositivos que tiene actualmente. Si ha aumentado o disminuido el número de ranuras, también verá el número de ranuras que tendrá un mes después de la fecha en que ha realizado el cambio.

Adquisición de una ranura de dispositivos (AWS CLI)

Puede ejecutar el comando purchase-offering para comprar la oferta.

Para publicar la configuración de su cuenta de Device Farm, incluido el número máximo de ranuras de dispositivos que puede adquirir y el número de minutos de evaluación gratuitos restantes de que dispone, ejecute el comando get-account-settings. Verá un resultado similar al siguiente:

```
{
    "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 obtener una lista de las ofertas de ranuras de dispositivos que tiene disponibles, ejecute el comando list-offerings. Debería ver una salida similar a esta:

```
{
    "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 mostrar la lista de promociones de ofertas disponibles, ejecute el comando list-offering-promotions.

> **Note**
>
> Este comando devuelve únicamente las promociones que aún no ha adquirido. En cuanto compre una o varias ranuras de alguna oferta mediante una promoción, dicha promoción dejará de aparecer en los resultados.

Debería ver una salida similar a esta:

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

Para obtener el estado de la oferta, ejecute el comando get-offering-status. Debería ver una salida similar a esta:

```
{
    "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
        }
    }
}
```

Los comandos renew-offering y list-offering-transactions también están disponibles para esta característica. Para obtener más información, consulte Referencia de AWS CLI.

Adquisición de una ranura de dispositivos (API)

1. Para mostrar la configuración de su cuenta, llame a la operación GetAccountSettings.
2. Para obtener una lista de las ofertas de ranuras de dispositivos disponibles, llame a la operación ListOfferings.
3. Para obtener una lista de las promociones de ofertas disponibles, llame a la operación ListOfferingPromotions.

> **Note**
>
> Este comando devuelve únicamente las promociones que aún no ha adquirido. En cuanto compre una o varias ranuras mediante una promoción de ofertas, dicha promoción dejará de aparecer en los resultados.

4. Para comprar una oferta, llame a la operación PurchaseOffering.
5. Para obtener el estado de la oferta, llame a la operación GetOfferingStatus.

Los comandos RenewOffering y ListOfferingTransactions también están disponibles para esta característica.

Para obtener más información acerca del uso de la API de Device Farm, consulte Automatización de Device Farm.

Conceptos de AWS Device Farm

En esta sección se describen conceptos importantes de Device Farm.

- Compatibilidad de dispositivos en AWS Device Farm
- Entornos de prueba
- Ejecuciones
- Informes en AWS Device Farm
- Sesiones

Para obtener más información acerca de los tipos de pruebas admitidos en Device Farm, consulte Trabajar con tipos de pruebas en AWS Device Farm.

Compatibilidad de dispositivos en AWS Device Farm

Las siguientes secciones contienen información sobre la compatibilidad con dispositivos en Device Farm.

Temas

- Dispositivos compatibles
- Grupos de dispositivos
- Dispositivos privados
- Marcas de dispositivos
- Ranuras de dispositivos
- Aplicaciones preinstaladas en los dispositivos
- Capacidades de los dispositivos

Dispositivos compatibles

Device Farm es compatible con cientos de dispositivos Android, iOS y Fire OS únicos y populares, y combinaciones de sistemas operativos. La lista de dispositivos disponibles crece a medida que se lanzan nuevos dispositivos al mercado. Para obtener lista completa de dispositivos, consulte Lista de dispositivos.

Grupos de dispositivos

Device Farm organiza sus dispositivos en grupos de dispositivos que puede utilizar para las pruebas. Estos grupos de dispositivos contienen dispositivos relacionados, como dispositivos que solo se ejecutan en Android o en iOS. Device Farm proporciona grupos de dispositivos preparados, como los formados por los principales dispositivos. También puede crear grupos de dispositivos que combinen dispositivos públicos y privados.

Dispositivos privados

Los dispositivos privados le permiten especificar configuraciones de hardware y software exactas para sus necesidades de pruebas. Algunas configuraciones, como los dispositivos Android rooteados, se pueden admitir como dispositivos privados. Cada dispositivo privado es un dispositivo físico que Device Farm implementa en su nombre en un centro de datos de Amazon. Sus dispositivos privados están a su exclusiva disposición, tanto para pruebas automáticas como manuales. Una vez que haya optado por finalizar su suscripción, el hardware se eliminará de nuestro entorno. Para obtener más información, consulte Dispositivos privados y Trabajar con dispositivos privados en AWS Device Farm.

Marcas de dispositivos

Device Farm realiza pruebas en dispositivos móviles y tabletas físicos de diversos fabricantes de equipos originales.

Ranuras de dispositivos

Las ranuras de dispositivos siguen un modelo de simultaneidad en el que el número de ranuras de dispositivos que ha adquirido determina cuántos dispositivos puede ejecutar en pruebas o en sesiones de acceso remoto.

Existen dos tipos de ranuras de dispositivos:

- Una ranura de dispositivos de acceso remoto es una ranura en la que puede ejecutar sesiones de acceso remoto de forma simultánea.

  Si tiene una ranura de dispositivos de acceso remoto, solo podrá ejecutar una sesión de acceso remoto a la vez. Si compra ranuras adicionales de dispositivos de acceso remoto, podrá ejecutar varias sesiones de forma simultánea.

- Una ranura de dispositivos de pruebas automatizadas es una ranura en el que se pueden ejecutar pruebas de forma simultánea.

  Si tiene una ranura de dispositivos de pruebas automatizadas, solo podrá ejecutar pruebas en un dispositivo a la vez. Si compra ranuras adicionales de dispositivos de pruebas automatizadas, podrá ejecutar varias pruebas simultáneamente en varios dispositivos para obtener los resultados de las pruebas más rápidamente.

Puede comprar ranuras de dispositivos en función de la familia de dispositivos (dispositivos Android o iOS para pruebas automatizadas y dispositivos Android o iOS para acceso remoto). Para obtener más información, consulte los precios de Device Farm.

Aplicaciones preinstaladas en los dispositivos

Los dispositivos en Device Farm incluyen un pequeño número de aplicaciones que ya han instalado los fabricantes o las operadoras.

Capacidades de los dispositivos

Todos los dispositivos disponen de conexión wifi con acceso a Internet. No tienen conexión al operador de servicios y no pueden hacer llamadas telefónicas ni enviar mensajes SMS.

Puede tomar fotos con cualquier dispositivo que admita una cámara frontal o trasera. Debido al modo en que están montados los dispositivos, las fotos podrían salir oscuras y borrosas.

Google Play Services está instalado en los dispositivos que lo soportan, pero estos dispositivos no tienen una cuenta de Google activa.

Entornos de prueba de AWS Device Farm

AWS Device Farm proporciona métricas personalizadas y entornos de pruebas estándar para ejecutar pruebas automatizadas. Puede elegir un entorno de pruebas personalizado para disponer de un control pleno de las pruebas automatizadas. Si lo prefiere, puede elegir el entorno de pruebas estándar predeterminado de Device Farm, que ofrece informes granulares de cada prueba del conjunto de pruebas automatizadas.

Temas

Entorno de pruebas estándar

Cuando se ejecuta una prueba en el entorno estándar, Device Farm proporciona registros detallados e informes para cada caso del conjunto de pruebas. Puede ver datos de desempeño, vídeos, capturas de pantalla y registros de cada prueba, con el fin de identificar y solucionar los errores de su aplicación.

> **Note**
>
> Dado que Device Farm ofrece informes granulares en el entorno estándar, las ejecuciones de pruebas pueden tardar más que cuando se ejecutan localmente. Si desea acortar los tiempos de ejecución, ejecute las pruebas en un entorno de pruebas personalizado.

Entorno de pruebas personalizado

Al personalizar el entorno de pruebas, puede especificar los comandos que Device Farm debe ejecutar para llevar a cabo las pruebas. De este modo, se garantiza que las pruebas se ejecuten en Device Farm de forma parecida a cuando se ejecutan en un equipo local. Ejecutar las pruebas en este modo también permite el streaming en directo de vídeo y de registros de las pruebas. Al ejecutar pruebas en un entorno de pruebas personalizado, no se obtienen informes granulares para cada caso de prueba. Para obtener más información, consulte Trabajo con entornos de pruebas personalizados.

Podrá elegir si desea utilizar un entorno de pruebas personalizado cuando utilice la consola de Device Farm, AWS CLI, o la API de Device Farm para crear una ejecución de prueba.

Para obtener más información, consulte Carga de una especificación de prueba personalizada conAWS CLI y Crear una ejecución de prueba en Device Farm.

Se ejecuta en AWS Device Farm

En las siguientes secciones se ofrece información sobre las ejecuciones en Device Farm.

Una ejecución en Device Farm representa una compilación específica en la aplicación, con un conjunto específico de pruebas, que se ejecutará en un conjunto específico de dispositivos. Una ejecución produce un informe que contiene información acerca de los resultados de la ejecución. Una ejecución contiene una o varias tareas.

Temas

Configuración de una ejecución

Como parte de una ejecución, puede suministrar ajustes que Device Farm puede usar para anular la configuración actual del dispositivo. Esto incluye coordenadas de latitud y longitud, configuración regional, estados de radio (como Bluetooth, GPS, NFC y wifi), datos adicionales (contenidos en un archivo .zip) y aplicaciones auxiliares (aplicaciones que deben estar instaladas antes que la aplicación que se va a probar).

Conservación de los archivos de las ejecuciones

Device Farm almacena las aplicaciones y los archivos durante 30 días y, a continuación, los elimina de su sistema. No obstante, puede eliminar los archivos en cualquier momento.

Device Farm almacena los resultados, los logs y las capturas de pantalla de las ejecuciones durante 400 días y, a continuación, los elimina de su sistema.

Estado del dispositivo en las ejecuciones

Device Farm siempre reinicia un dispositivo antes de que esté disponible para la siguiente tarea.

Ejecuciones en paralelo

Device Farm ejecuta pruebas en paralelo a medida que los dispositivos van estando disponibles.

Configuración del tiempo de espera de ejecución

Puede establecer un valor por el tiempo durante el cual se debería llevar a cabo una ejecución de prueba antes de detener la ejecución de una prueba en cada uno de los dispositivos. Por ejemplo, si las pruebas tardan 20 minutos en completarse por dispositivo, debe elegir un tiempo de espera de 30 minutos por dispositivo.

Para obtener más información, consulte Establecimiento del tiempo de espera de ejecución para ejecuciones de pruebas en AWS Device Farm.

Instrumentación de aplicaciones

No es necesario instrumentar sus aplicaciones ni proporcionar a Device Farm el código fuente para las aplicaciones. Las aplicaciones Android se pueden enviar sin modificar. Las aplicaciones iOS deben crearse con Dispositivo iOS como destino en lugar un simulador.

Volver a firmar las aplicaciones en las ejecuciones

Para las aplicaciones iOS, no necesita añadir ningún UUID de Device Farm al perfil de aprovisionamiento. Device Farm sustituye el perfil de aprovisionamiento integrado por un perfil comodín y, a continuación, vuelve a firmar la aplicación. Si proporciona datos auxiliares, Device Farm los añade al paquete de la aplicación antes de que Device Farm la instale, de modo que esos datos estén presentes en el entorno aislado de la aplicación. Al volver a firmar la aplicación, se eliminan derechos como el grupo de aplicaciones, los dominios asociados, el Game Center, la configuración de accesorios inalámbricos HealthKit HomeKit, la compra integrada en la aplicación, el audio entre aplicaciones, Apple Pay, las notificaciones push y la configuración y el control de la VPN.

Para aplicaciones Android, Device Farm vuelve a firmar la aplicación. Esto podría interrumpir cualquier funcionalidad que dependa de la firma de la aplicación, como la API de Google Maps para Android, o podría activar la detección antipiratería o antimanipulación por parte de productos como. DexGuard

Aplicaciones ocultas en las ejecuciones

En el caso de las aplicaciones de Android, si la aplicación está ofuscada, puedes probarla con Device Farm si la usas. ProGuard Sin embargo, si la utilizas DexGuard con medidas antipiratería, Device Farm no puede volver a firmar ni realizar pruebas con la aplicación.

Anuncios en las ejecuciones

Le aconsejamos que elimine los anuncios de las aplicaciones antes de cargarlas en Device Farm. No podemos garantizar que se muestren los anuncios durante las ejecuciones.

Medios en las ejecuciones

Puede proporcionar medios u otros datos para acompañar a su aplicación. Los datos adicionales se deben proporcionar en un archivo.zip de hasta 4 GB de tamaño.

Tareas comunes para ejecuciones

Para obtener más información, consulte Crear una ejecución de prueba en Device Farm y Trabajo con ejecuciones de prueba en AWS Device Farm.

Informes en AWS Device Farm

En las siguientes secciones, se ofrece información acerca de los informes de las pruebas de Device Farm.

Temas

- Conservación de informes
- Componentes de informes
- Registros en informes
- Tareas comunes para informes

Conservación de informes

Device Farm almacena sus informes durante 400 días. Estos informes incluyen metadatos, logs, capturas de pantalla y datos de desempeño.

Componentes de informes

Los informes en Device Farm contienen información de éxitos y errores, informes de errores, registros de pruebas y dispositivos, capturas de pantalla y datos de desempeño.

Los informes contienen datos detallados por dispositivo, así como resultados generales, como el número de veces que se ha producido un determinado problema.

Registros en informes

Los informes incluyen capturas de logcat completas para pruebas de Android y registros completos de la consola de dispositivos para pruebas de iOS.

Tareas comunes para informes

Para obtener más información, consulte Trabajar con informes de pruebas en Device Farm.

Sesiones de AWS Device Farm

Puede utilizar Device Farm para realizar pruebas interactivas de aplicaciones Android e iOS a través de sesiones de acceso remoto en un navegador web. Este tipo de pruebas interactivas ayuda a los ingenieros de soporte, durante una llamada con el cliente, a recorrer paso a paso el problema del cliente. Los desarrolladores pueden reproducir un problema en un dispositivo concreto para aislar posibles orígenes del problema. Puede utilizar las sesiones remotas para realizar pruebas de facilidad de uso con sus clientes objetivo.

Temas

Dispositivos compatibles con el acceso remoto

Device Farm es compatible con una serie de dispositivos Android e iOS únicos y populares. La lista de dispositivos disponibles crece a medida que se lanzan nuevos dispositivos al mercado. En la consola de Device Farm se muestra la lista actual de dispositivos Android e iOS disponibles para el acceso remoto. Para obtener más información, consulte Compatibilidad de dispositivos en AWS Device Farm.

Conservación de los archivos de sesión

Device Farm almacena las aplicaciones y los archivos durante 30 días y, a continuación, los elimina de su sistema. No obstante, puede eliminar los archivos en cualquier momento.

Device Farm almacena los registros y el vídeo capturado en la sesión durante 400 días y, a continuación, los elimina de su sistema.

Instrumentación de aplicaciones

No es necesario instrumentar sus aplicaciones ni proporcionar a Device Farm el código fuente para las aplicaciones. Las aplicaciones Android e iOS se pueden enviar sin modificar.

Volver a firmar las aplicaciones en las sesiones

Device Farm vuelve a firmar las aplicaciones Android e iOS. Esto puede interrumpir las funcionalidades que dependan de la firma de la aplicación. Por ejemplo, la API de Google Maps para Android depende de la firma de su aplicación. La nueva firma de aplicaciones también podría activar los sistemas antipiratería y antimanipulación disponibles en productos como DexGuard para dispositivos Android.

Aplicaciones ocultas en las sesiones

Para aplicaciones Android, si la aplicación está oculta, aún puede probarla con Device Farm si utiliza ProGuard. Sin embargo, si utiliza DexGuard con medidas antipiratería, Device Farm no puede volver a firmar la aplicación.

Trabajar con proyectos en AWS Device Farm

Un proyecto en Device Farm representa un espacio de trabajo lógico que contiene ejecuciones, una ejecución para cada prueba de una sola aplicación contra uno o más dispositivos. Los proyectos le permiten organizar espacios de trabajo de la forma que elija. Por ejemplo, puede haber un proyecto por título de aplicación, o puede haber un proyecto por plataforma. Puede crear todos los proyectos que necesite.

Puede usar la consola AWS Device Farm, AWS Command Line Interface (AWS CLI) o la API de AWS Device Farm para trabajar con proyectos.

Temas

Crear un proyecto en AWS Device Farm

Puede crear un proyecto utilizando la consola de AWS Device Farm, la AWS CLI o la API de AWS Device Farm.

Requisitos previos

- Realice los pasos que se indican en Configuración.

Crear un proyecto (consola)

1. Inicie sesión en la consola de Device Farm en https://console.aws.amazon.com/devicefarm.
2. En el panel de navegación de Device Farm, seleccione Pruebas de dispositivos móviles y, a continuación, seleccione Proyectos.
3. Seleccione Nuevo proyecto.
4. Introduzca un nombre para el proyecto y, a continuación, seleccione Enviar.
5. Para especificar la configuración del proyecto, seleccione Configuración del proyecto. En esta configuración se incluye el tiempo de espera predeterminado para las ejecuciones de prueba.

Una vez que se ha aplicado la configuración, se utiliza en todas las ejecuciones de prueba del proyecto. Para obtener más información, consulte Establecimiento del tiempo de espera de ejecución para ejecuciones de pruebas en AWS Device Farm.

Crear un proyecto (AWS CLI)

- Ejecute create-project especificando el nombre del proyecto.

  Ejemplo:

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

  La respuesta de la AWS CLI incluye el nombre de recurso de Amazon (ARN) del proyecto.

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

  Para obtener más información, consulte create-project y Referencia de AWS CLI.

Crear un proyecto (API)

- Llame a la API `CreateProject`.

Para obtener más información acerca del uso de la API de Device Farm, consulte Automatización de Device Farm.

Visualizar la lista de proyectos en AWS Device Farm

Puede usar la consola de AWS Device Farm, AWS CLI, o la API de AWS Device Farm para ver la lista de proyectos.

Temas

Requisitos previos

- Cree al menos un proyecto en Device Farm. Siga las instrucciones de Crear un proyecto en AWS Device Farm y, a continuación, vuelva a esta página.

Visualización de la lista de proyectos (consola)

1. Inicie sesión en la consola de Device Farm en https://console.aws.amazon.com/devicefarm.
2. Para buscar la lista de proyectos disponibles, haga lo siguiente:
   - Para proyectos de pruebas de dispositivos móviles, en el menú de navegación de Device Farm, seleccione Pruebas de dispositivos móviles y, a continuación, seleccione Proyectos.
   - Para los proyectos de pruebas de navegadores de escritorio, en el menú de navegación de Device Farm, seleccione Pruebas de navegadores de escritorio y, a continuación, seleccione Proyectos.

Visualizar la lista de proyectos (AWS CLI)

- Para ver la lista de proyectos, ejecute el comando list-projects.

  Para ver información sobre un único proyecto, ejecute el comando get-project.

Para obtener más información sobre el uso de Device Farm con la AWS CLI, consulte Referencia de AWS CLI.

Visualizar la lista de proyectos (API)

- Para ver la lista de proyectos, llame a la API `ListProjects`.

  Para ver información sobre un único proyecto, llame a la API `GetProject`.

Para obtener información sobre la API AWS Device Farm, consulte Automatización de Device Farm.

Trabajo con ejecuciones de prueba en AWS Device Farm

Una ejecución en Device Farm representa una compilación específica en la aplicación, con un conjunto específico de pruebas, que se ejecutará en un conjunto específico de dispositivos. Una ejecución produce un informe que contiene información acerca de los resultados de la ejecución. Una ejecución contiene una o varias tareas. Para obtener más información, consulte Ejecuciones.

Puede usar la consola AWS Device Farm, AWS Command Line Interface (AWS CLI) o la API AWS Device Farm para trabajar con las ejecuciones.

Temas

Crear una ejecución de prueba en Device Farm

Puede usar la consola Device Farm o la API de Device Farm para crear una ejecución de prueba. AWS CLI También puede utilizar un complemento admitido, como, por ejemplo, los complementos Jenkins o Gradle para Device Farm. Para obtener más información acerca de los complementos, consulte Herramientas y complementos. Para obtener información acerca de las ejecuciones, consulte Ejecuciones.

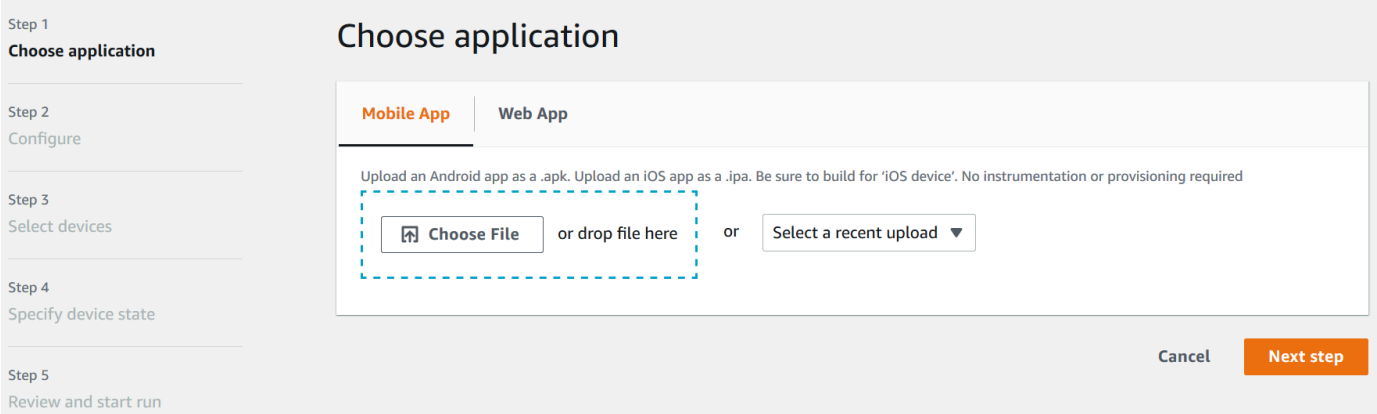
Temas

Requisitos previos

Debe tener un proyecto en Device Farm. Siga las instrucciones de Crear un proyecto en AWS Device Farm y, a continuación, vuelva a esta página.

Creación de una ejecución de prueba (consola)

1. Inicie sesión en la consola de Device Farm en https://console.aws.amazon.com/devicefarm.
2. En el panel de navegación, seleccione Pruebas de dispositivos móviles y, a continuación, seleccione Proyectos.
3. Si ya dispone de un proyecto, puede cargar las pruebas en él. En caso contrario, seleccione Nuevo proyecto, indique un Nombre del proyecto y seleccione Crear.
4. Abra el proyecto y, a continuación, seleccione Crear una nueva ejecución.
5. En la página Elija su aplicación, seleccione Aplicación móvil o Aplicación web.



6. Cargue el archivo de la aplicación. También puede arrastrar y soltar el archivo o elegir una carga reciente. Si carga una aplicación iOS, asegúrese de elegir Dispositivo iOS, en lugar de un simulador.
7. (Opcional) En Nombre de la ejecución, escriba un nombre. De forma predeterminada, Device Farm usa el nombre del archivo de la aplicación.
8. Seleccione Siguiente.
9. En la página Configurar, elija uno de los conjuntos de pruebas disponibles.

> **Note**
>
> Si no hay ninguna prueba disponible, elija Integrado: fuzzing para ejecutar un conjunto de pruebas integrado estándar. Si elige Integrado: fuzzing y aparecen los cuadros

Recuento de eventos, Acelerador de eventos y Semilla aleatorizadora, puede cambiar los valores o conservarlos como están.

Para obtener más información acerca de los conjuntos de pruebas disponibles, consulte Trabajar con tipos de pruebas en AWS Device Farm.

10. Si no ha seleccionado Integrado: fuzzing, seleccione Elegir archivo. A continuación, busque y seleccione el archivo que contiene las pruebas.
11. Para su entorno de prueba, seleccione Ejecutar la prueba en nuestro entorno estándar o Ejecutar la prueba en un entorno personalizado. Para obtener más información, consulte Entornos de prueba.
12. Si utiliza el entorno de pruebas estándar, vaya directamente al paso 13. Si utiliza un entorno de pruebas personalizado con el archivo YAML de especificación de prueba predeterminado, vaya directamente al paso 13.

    a. Si desea editar la especificación de prueba predeterminada en un entorno de pruebas personalizado, seleccione Editar para actualizar la especificación YAML predeterminada.

    b. Si ha modificado la especificación de prueba, seleccione Guardar como nuevo para actualizarla.
13. Si desea configurar las opciones de grabación de vídeo o captura de datos de desempeño, elija Configuración avanzada.

    a. Seleccione Habilitar grabación de vídeo para grabar vídeo durante la prueba.

    b. Seleccione Habilitar la captura de datos de rendimiento de aplicaciones para capturar datos de desempeño en el dispositivo.

> **Note**
>
> Si tiene dispositivos privados, se muestra también la opción Configuración específica para dispositivos privados.

14. Seleccione Siguiente.
15. En la página Seleccionar dispositivos, realice una de las siguientes opciones:

    - Para elegir un grupo de dispositivos integrados donde ejecutar las pruebas, en Grupo de dispositivos, elija Dispositivos principales.

- Para crear su propio grupo de dispositivos donde ejecutar las pruebas, siga las instrucciones de Crear un grupo de dispositivos y, a continuación, regrese a esta página.
- Si ha creado su propio grupo de dispositivos antes, en Grupo de dispositivos, elija su grupo de dispositivos.

Para obtener más información, consulte Compatibilidad de dispositivos en AWS Device Farm.

16. Seleccione Siguiente.
17. En la página Especificar el estado del dispositivo:

    - Para proporcionar otros datos para que Device Farm los utilice durante la ejecución, junto a Agregar datos adicionales, seleccione Elegir archivo y, a continuación, busque el archivo .zip que contiene los datos.
    - Para instalar una aplicación adicional que Device Farm utilizará durante la ejecución, junto a Instalar otras aplicaciones, seleccione Elegir archivo y, a continuación, busque y seleccione el archivo .apk o .ipa que contiene la aplicación. Repita la acción para las demás aplicaciones que desee instalar. Puede cambiar el orden de instalación arrastrando y soltando las aplicaciones después de cargarlas.
    - Para especificar si las opciones de wifi, Bluetooth, GPS o NFC estarán habilitadas durante la ejecución, junto a Definir estados de radio, seleccione las casillas correspondientes.
    - Para preestablecer la latitud y la longitud del dispositivo para la ejecución, junto a Ubicación del dispositivo, escriba las coordenadas.
    - Para preestablecer la configuración regional del dispositivo para la ejecución, seleccione la configuración regional en Configuración regional del dispositivo.
18. Seleccione Siguiente.
19. Cuando llega a Revisar e iniciar ejecución, puede especificar el tiempo de espera de ejecución de la prueba. Si utiliza un número ilimitado de ranuras de pruebas, confirme que la opción Ejecutar en ranuras no contabilizadas esté seleccionada.
20. Escriba un valor o utilice la barra del control deslizante para modificar el tiempo de espera de la ejecución. Para obtener más información, consulte Establecimiento del tiempo de espera de ejecución para ejecuciones de pruebas en AWS Device Farm.
21. Seleccione Confirmar e iniciar ejecución.

Device Farm comenzará la ejecución tan pronto como los dispositivos estén disponibles, normalmente en unos minutos. Durante la ejecución

de la prueba, la consola de Device Farm mostrará un icono pendiente

en la tabla de ejecución. Cada dispositivo en ejecución también empezará con el icono de pendiente y, después, pasará al icono de ejecución

cuando comience la prueba. Al finalizar cada prueba, aparece un icono con el resultado de la prueba junto al nombre del dispositivo. Cuando se hayan completado todas las pruebas, el icono de pendientes situado junto a la ejecución pasará a ser el icono del resultado de la prueba.

Si necesita detener la ejecución de prueba, consulte Detener una ejecución en AWS Device Farm.

Creación de una ejecución de prueba (AWS CLI)

Puede utilizarla AWS CLI para crear una ejecución de prueba.

Temas

- Paso 1: Elegir un proyecto
- Paso 2: Elegir un grupo de dispositivos
- Paso 3: Cargar el archivo de la aplicación
- Paso 4: Cargar el paquete de scripts de pruebas
- Paso 5: Cargar la especificación de prueba personalizada (opcional)
- Paso 6: Programar una ejecución de prueba

Paso 1: Elegir un proyecto

Debe asociar la ejecución de prueba a un proyecto de Device Farm.

1. Para ver una lista de sus proyectos de Device Farm, ejecute list-projects. Si no dispone de ningún proyecto, consulte Crear un proyecto en AWS Device Farm.

   Ejemplo:

   ```
   aws devicefarm list-projects
   ```

   La respuesta incluye una lista de proyectos de Device Farm.

   ```
   {
   ```

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

2. Seleccione un proyecto para asociarlo a la ejecución de prueba y anote su nombre de recurso de Amazon (ARN).

Paso 2: Elegir un grupo de dispositivos

Debe elegir un grupo de dispositivos para asociárselo a la ejecución de prueba.

1. Para ver los grupos de dispositivos, ejecute list-device-pools especificando el ARN del proyecto.

   Ejemplo:

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

   La respuesta incluye los grupos de dispositivos integrados de Device Farm, tales como Top Devices, así como todos los grupos de dispositivos creados previamente para este proyecto:

   ```
   {
       "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. Elija un grupo de dispositivos y anote su ARN.

   También puede crear un grupo de dispositivos y, a continuación, volver a este paso. Para obtener más información, consulte Crear un grupo de dispositivos (AWS CLI).

Paso 3: Cargar el archivo de la aplicación

Para crear la solicitud de carga y obtener una URL de carga prefirmada de Amazon Simple Storage Service (Amazon S3), necesita lo siguiente:

- El ARN de su proyecto.
- El nombre del archivo de aplicación.
- El tipo de carga.

Para obtener más información, consulte create-upload.

1. Para cargar un archivo, ejecute create-upload con los parámetros `--project-arn`, `--name` y `--type`.

   En este ejemplo se crea una carga para una aplicación Android:

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

La respuesta incluye el ARN de carga de la aplicación y una URL prefirmada.

```
{
    "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 el ARN de carga de la aplicación y la URL prefirmada.
3. Cargar el archivo de la aplicación mediante la URL prefirmada de Amazon S3. En este ejemplo se utiliza curl para cargar un archivo .apk de Android:

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

Para obtener más información, consulte Carga de objetos con URL prefirmadas en la Guía para usuarios de Amazon Simple Storage Service.

4. Para comprobar el estado de la carga de la aplicación, ejecute get-upload y especifique el ARN de carga de la aplicación.

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

Espere hasta que el estado contenido en la respuesta sea SUCCEEDED antes de cargar el paquete de scripts de pruebas.

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

Paso 4: Cargar el paquete de scripts de pruebas

A continuación, cargue el paquete de scripts de pruebas.

1. Para crear la solicitud de carga y obtener una URL de carga prefirmada de Amazon S3, ejecute create-upload con los parámetros `--project-arn`, `--name` y `--type`.

   En este ejemplo se crea una carga de paquete de pruebas de Appium Java TestNG:

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

   La respuesta incluye el ARN de carga del paquete de pruebas y una URL prefirmada.

   ```
   {
       "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 el ARN de carga del paquete de pruebas y la URL prefirmada.
3. Cargue el archivo del paquete de scripts de pruebas mediante la URL prefirmada de Amazon S3. En este ejemplo se utiliza curl para cargar un archivo comprimido de scripts de Appium TestNG:

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

4. Para comprobar el estado de la carga del paquete de scripts de pruebas, ejecute get-upload y especifique el ARN de carga del paquete de pruebas del paso 1.

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

Espere a que el estado contenido en la respuesta sea SUCCEEDED antes de continuar al paso siguiente, que es 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}"
    }
}
```

Paso 5: Cargar la especificación de prueba personalizada (opcional)

Si utiliza las pruebas en un entorno de pruebas estándar, omita este paso.

Device Farm mantiene un archivo de especificación de prueba predeterminado para cada tipo de prueba admitido. A continuación, descargue la especificación de prueba predeterminada y utilícela para crear una carga de especificación de prueba personalizada con el fin de ejecutar las pruebas en un entorno de pruebas personalizado. Para obtener más información, consulte Entornos de prueba.

1. Para encontrar el ARN de carga de la especificación de prueba predeterminada, ejecute list-uploads y especifique el ARN del proyecto.

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

La respuesta contiene una entrada para cada especificación de prueba predeterminada:

```
{
    "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. Seleccione la especificación de prueba predeterminada de la lista. Anote su ARN de carga.
3. Para descargar la especificación de prueba predeterminada, ejecute get-upload y especifique el ARN de carga.

   Ejemplo:

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

   La respuesta contiene una URL prefirmada en la que podrá descargar la especificación de prueba predeterminada.
4. En este ejemplo se utiliza curl para descargar la especificación de prueba predeterminada y guardarla como `MyTestSpec.yml`:

   ```
   curl "https://prod-us-west-2-uploads.s3-us-west-2.amazonaws.com/ExampleURL" >
    MyTestSpec.yml
   ```
5. Puede editar la especificación de prueba predeterminada de tal forma que satisfaga sus requisitos de pruebas y, a continuación, utilizar la especificación de prueba modificada en futuras ejecuciones de prueba. Omita este paso si desea usar la especificación de prueba predeterminada tal cual en un entorno de pruebas personalizado.

6. Para crear una carga de la especificación de prueba personalizada, ejecute create-upload especificando el nombre de la especificación de prueba, su tipo y el ARN del proyecto.

   En este ejemplo se crea una carga para una especificación de prueba personalizada de Appium Java TestNG:

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

   La respuesta incluye el ARN de carga de la especificación de prueba y la URL prefirmada:

   ```
   {
       "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 el ARN de carga de la especificación de prueba y la URL prefirmada.

8. Cargue el archivo de la especificación de prueba mediante la URL prefirmada de Amazon S3. Este ejemplo se utiliza curl para cargar una especificación de prueba de Appium JavaTest NG:

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

9. Para comprobar el estado de la carga de la especificación de prueba, ejecute get-upload y especifique el ARN de carga.

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

   Espere hasta que el estado contenido en la respuesta sea SUCCEEDED antes de programar la ejecución de prueba.

```
{
    "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 actualizar la especificación de prueba personalizada, ejecute update-upload especificando el ARN de carga de la especificación de prueba. Para obtener más información, consulte update-upload.

Paso 6: Programar una ejecución de prueba

Para programar una ejecución de prueba con AWS CLI, ejecuteschedule-run, especificando:

- El ARN del proyecto del paso 1.
- El ARN del grupo de dispositivos del paso 2.
- El ARN de carga de la aplicación del paso 3.
- El ARN de carga del paquete de prueba del paso 4.

Si ejecuta las pruebas en un entorno de pruebas personalizado, también necesita el ARN de la especificación de prueba del paso 5.

Para programar una ejecución en un entorno de pruebas estándar

- Ejecute schedule-run especificando el ARN del proyecto, el ARN del grupo de dispositivos, el ARN de carga de la aplicación y la información del paquete de pruebas.

  Ejemplo:

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

La respuesta contiene un ARN de ejecución que puede utilizar para comprobar el estado de la ejecución de prueba.

```
{
    "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 obtener más información, consulte schedule-run.

Para programar una ejecución en un entorno de pruebas personalizado

- Los pasos son prácticamente los mismos que para el entorno de pruebas estándar, pero se incluye un atributo `testSpecArn` adicional incluido en el parámetro `--test`.

  Ejemplo:

```
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 comprobar el estado de la ejecución de prueba

- Utilice el comando get-run y especifique el ARN de la ejecución:

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

Para obtener más información, consulte get-run. Para obtener información sobre el uso de Device Farm con AWS CLI, consulteReferencia de AWS CLI.

Creación de una ejecución de prueba (API)

Los pasos son los mismos que los descritos en la AWS CLI sección. Consulte Creación de una ejecución de prueba (AWS CLI).

Para llamar a la API `ScheduleRun`, se requiere la información siguiente:

- Un ARN de proyecto. Consulte Crear un proyecto (API) y `CreateProject`.
- Un ARN de carga de una aplicación. Consulte `CreateUpload`.
- Un ARN de carga de paquete de pruebas. Consulte `CreateUpload`.

- Un ARN de grupos de dispositivos. Consulte Crear un grupo de dispositivos y `CreateDevicePool`.

> **Note**
>
> Si ejecuta las pruebas en un entorno de pruebas personalizado, también necesita el ARN de carga de la especificación de prueba. Para obtener más información, consulte Paso 5: Cargar la especificación de prueba personalizada (opcional) y `CreateUpload`.

Para obtener más información acerca del uso de la API de Device Farm, consulte Automatización de Device Farm.

Siguientes pasos

En la consola de Device Farm, el icono de reloj

se convertirá en un icono de resultado, como el icono de ejecución correcta

, cuando se complete la ejecución. Tan pronto como se completan las pruebas, aparece un informe de la ejecución. Para obtener más información, consulte Informes en AWS Device Farm.

Para usar el informe, siga las instrucciones que se indican en Trabajar con informes de pruebas en Device Farm.

Establecimiento del tiempo de espera de ejecución para ejecuciones de pruebas en AWS Device Farm

Puede establecer un valor por el tiempo durante el cual se debería llevar a cabo una ejecución de prueba antes de detener la ejecución de una prueba en cada uno de los dispositivos. El tiempo de espera de ejecución predeterminado es de 150 minutos por dispositivo, pero puede establecer un valor de tan solo 5 minutos. Puede usar la consola de AWS Device Farm o la API de AWS Device Farm para configurar el tiempo de espera de la ejecución. AWS CLI

> **Important**
>
> La opción de tiempo de espera de ejecución se debe establecer en la duración máxima de una ejecución de prueba, con cierto tiempo de reserva. Por ejemplo, si las pruebas tardan 20 minutos por dispositivo, debe elegir un tiempo de espera de 30 minutos por dispositivo.

Si la ejecución supera el tiempo de espera, se forzará la parada de la ejecución en ese dispositivo. Estarán disponibles los resultados parciales, si es posible. Se le facturará por la ejecución hasta ese momento, si está utilizando la opción de facturación con medidor. Para obtener más información, consulte los precios de Device Farm.

Puede que desee utilizar esta característica si sabe cuánto tiempo se supone que tardará en llevase a cabo una ejecución de prueba en cada dispositivo. Al especificar el tiempo de espera de una ejecución de prueba, puede evitar la situación en la que una ejecución de prueba se atasca por algún motivo y usted sigue siendo facturado por minutos de dispositivo aunque no haya ninguna prueba en ejecución. En otras palabras, el uso de la característica del tiempo de espera de ejecución le permite parar la ejecución de prueba si esta tarda más de lo previsto.

Puede establecer el tiempo de espera de ejecución en dos lugares: en el nivel del proyecto y en el nivel de la ejecución de prueba.

Requisitos previos

1. Realice los pasos que se indican en Configuración.
2. Cree un proyecto en Device Farm. Siga las instrucciones de Crear un proyecto en AWS Device Farm y, a continuación, vuelva a esta página.

Establecimiento del tiempo de espera de ejecución para un proyecto

1. Inicie sesión en la consola de Device Farm en https://console.aws.amazon.com/devicefarm.
2. En el panel de navegación de Device Farm, seleccione Pruebas de dispositivos móviles y, a continuación, seleccione Proyectos.
3. Si ya tiene un proyecto, selecciónelo de la lista. De lo contrario, seleccione Nuevo proyecto, introduzca un nombre para el proyecto y, a continuación, seleccione Enviar.
4. Seleccione Configuración del proyecto.

5. En la pestaña General, en Tiempo de espera de ejecución, escriba un valor o use la barra del control deslizante.
6. Seleccione Guardar.

   Ahora, todas las ejecuciones de prueba del proyecto usarán el valor de tiempo de espera de ejecución que ha especificado, a menos que anule el valor de tiempo de espera al programar una ejecución.

Establecimiento del tiempo de espera de ejecución para una ejecución de prueba

1. Inicie sesión en la consola de Device Farm en https://console.aws.amazon.com/devicefarm.
2. En el panel de navegación de Device Farm, seleccione Pruebas de dispositivos móviles y, a continuación, seleccione Proyectos.
3. Si ya tiene un proyecto, selecciónelo de la lista. De lo contrario, seleccione Nuevo proyecto, introduzca un nombre para el proyecto y, a continuación, seleccione Enviar.
4. Seleccione Crear una nueva ejecución.
5. Siga los pasos para elegir una aplicación, configurar la prueba, seleccionar los dispositivos y especificar el estado de un dispositivo.
6. En la pestaña Revisar e iniciar ejecución, en Establecer el tiempo de espera de ejecución, escriba un valor o use la barra del control deslizante.
7. Seleccione Confirmar e iniciar ejecución.

Simulación de conexiones y condiciones de una de red para ejecuciones de AWS Device Farm

Puede usar una forma de red para simular las conexiones y las condiciones de una red mientras realiza pruebas de las aplicaciones de Android, iOS, FireOS y web en Device Farm. Por ejemplo, puede probar la aplicación con condiciones de red inferiores a las ideales.

Cuando se crea una ejecución usando la configuración de red predeterminada, todos los dispositivos tienen una conexión wifi libre y completa con acceso a Internet. Cuando utiliza el modelado de red, puede cambiar la conexión Wi-Fi para especificar un perfil de red, como 3G o con pérdida, WiFi

que controle el rendimiento, el retraso, la fluctuación y las pérdidas tanto del tráfico entrante como saliente.

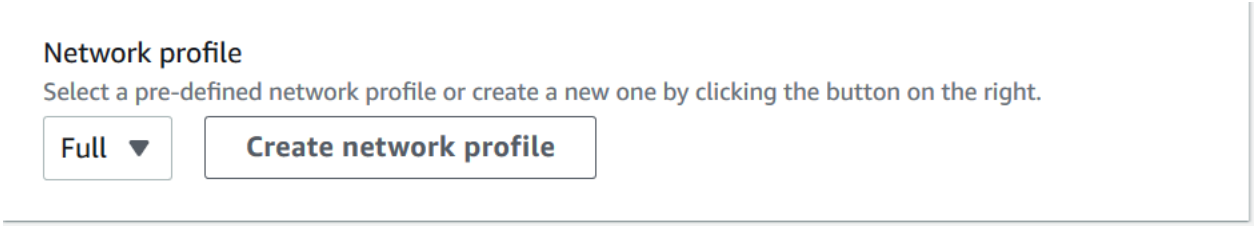
Temas

Configuración de la forma de red al programar una ejecución de prueba

Cuando programe una ejecución, puede elegir cualquiera de los perfiles mantenidos por Device Farm o bien puede crear y administrar el suyo propio.

1. Desde cualquier proyecto de Device Farm, seleccione Crear una nueva ejecución.

   Si aún no tiene ningún proyecto, consulte Crear un proyecto en AWS Device Farm.
2. Elija la aplicación y, a continuación, seleccione Siguiente.
3. Configure la prueba y, a continuación, seleccione Siguiente.
4. Seleccione los dispositivos y, a continuación, seleccione Siguiente.
5. En la sección Configuración de ubicación y red, seleccione un perfil de red o seleccione Crear perfil de red para crear el suyo propio.

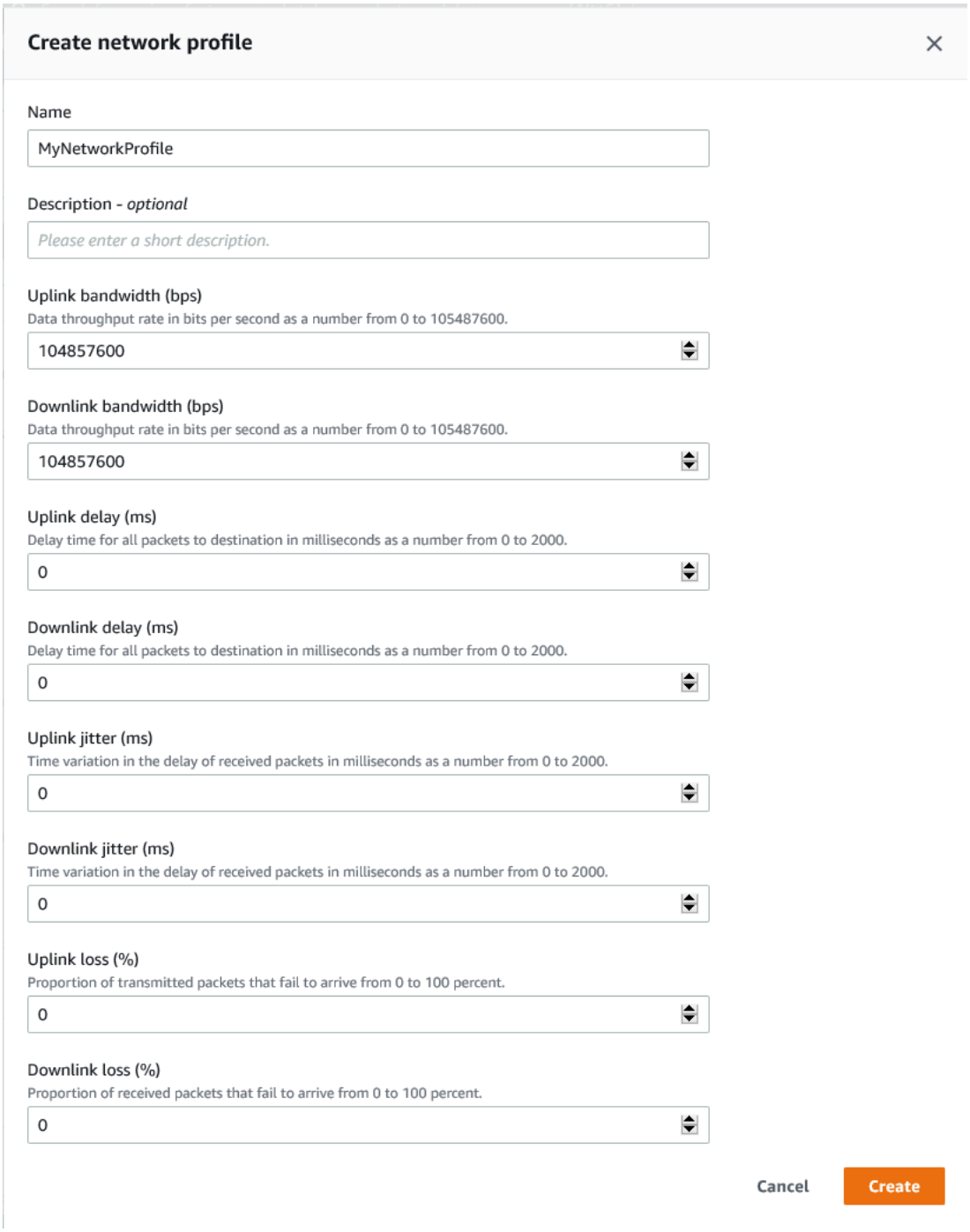


6. Seleccione Siguiente.
7. Revise y comience la ejecución de prueba.

Creación de un perfil de red

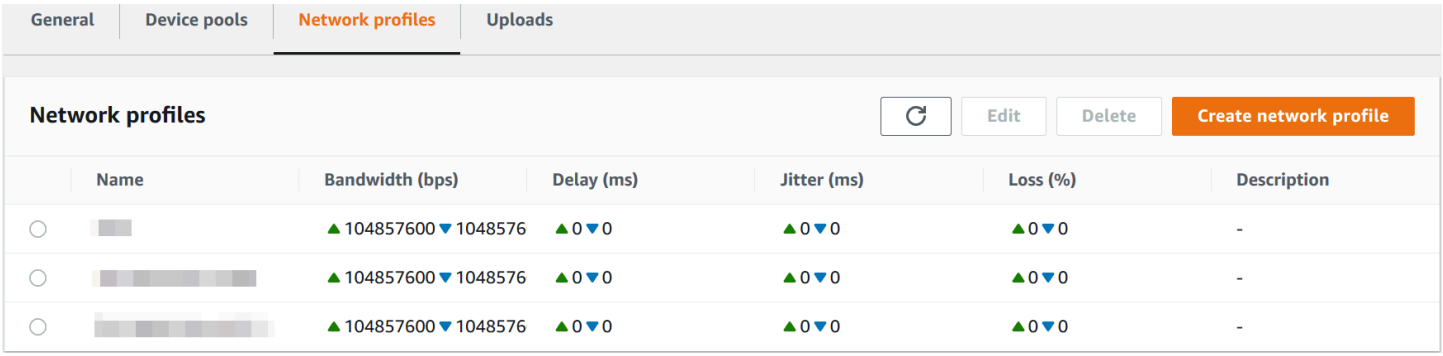
Al crear una ejecución de prueba, se puede crear un perfil de red.

1. Seleccione Crear un nuevo perfil de red.



2. Escriba un nombre y la configuración del perfil de red.
3. Seleccione Crear.
4. Finalice la creación de la ejecución de prueba y comience la ejecución.

Una vez creado el perfil de red, podrá verlo y administrarlo en la página Configuración del proyecto.



Cambio de las condiciones de red durante la prueba

Puede llamar a una API desde su host de dispositivos mediante un marco como Appium, con el fin de simular condiciones de red dinámicas como un ancho de banda reducido durante la ejecución de prueba. Para obtener más información, consulte. CreateNetworkProfile

Detener una ejecución en AWS Device Farm

Es posible que desee parar una ejecución después de haberla iniciado. Por ejemplo, si observa un problema mientras está ejecutando las pruebas, puede que desee reiniciar la ejecución con un script de prueba actualizado.

Puedes usar la consola o la API de AWS CLI Device Farm para detener una ejecución.

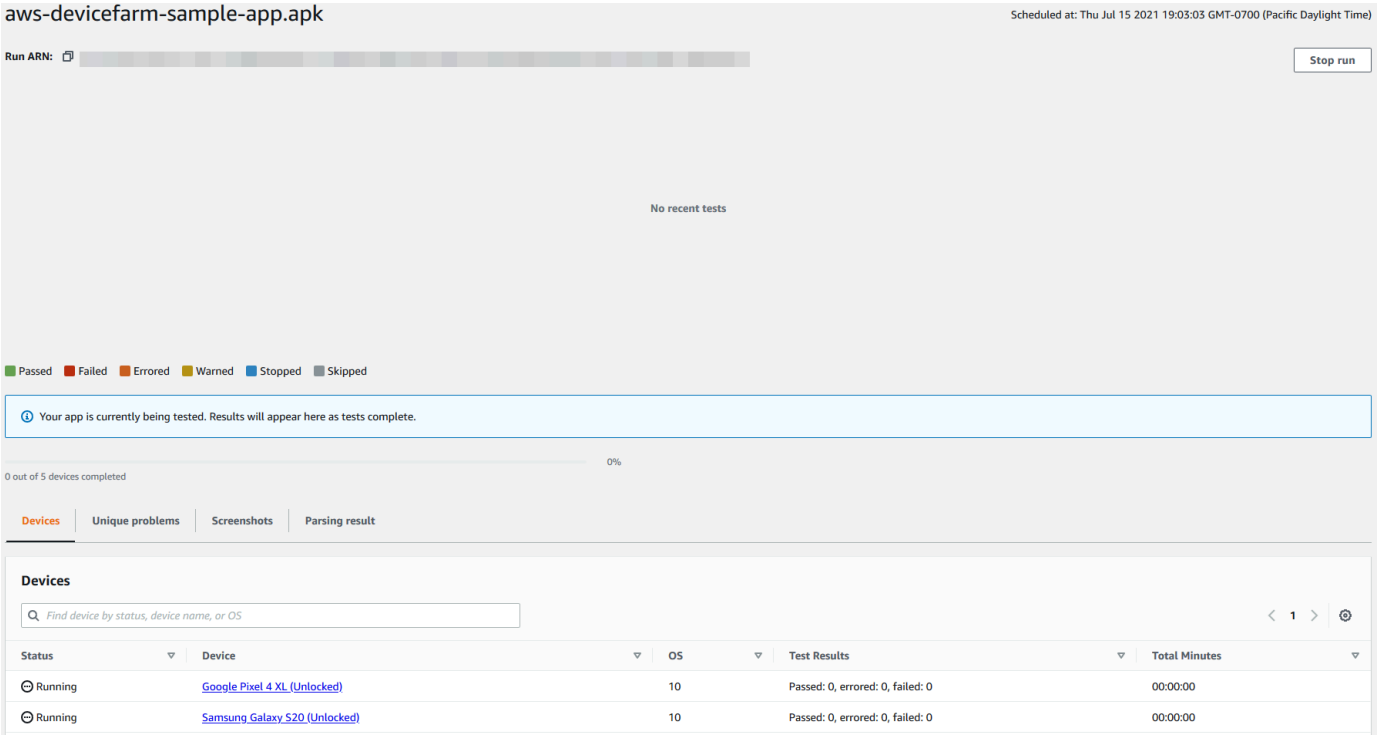
Temas

- Parar una ejecución (consola)
- Detener una ejecución (AWS CLI)
- Detener una ejecución (API)

Parar una ejecución (consola)

1. Inicie sesión en la consola de Device Farm en https://console.aws.amazon.com/devicefarm.
2. En el panel de navegación de Device Farm, seleccione Pruebas de dispositivos móviles y, a continuación, seleccione Proyectos.
3. Seleccione el proyecto en el que tiene una ejecución de prueba activa.
4. En la página Pruebas automatizadas, seleccione la ejecución de prueba.

El icono de ejecución pendiente o en curso debe aparecer a la izquierda del nombre del dispositivo.

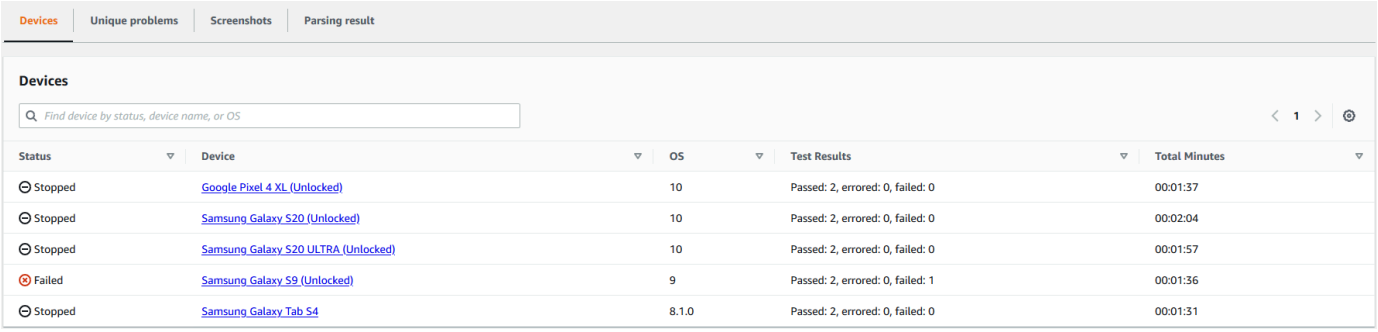


5. Seleccione Detener ejecución.

Transcurrido un breve periodo de tiempo, junto al nombre de dispositivo aparece un icono con un círculo rojo con un símbolo menos en su interior. Cuando se detiene la ejecución, el color del icono cambia de rojo a negro.

> **Important**
>
> Si una prueba ya se ha ejecutado, Device Farm no puede detenerla. Si una prueba está en curso, Device Farm la detiene. El total de minutos que se le facturarán aparece en la sección Dispositivos. Además, se le facturará el total de minutos que Device Farm tarde en ejecutar el conjunto de configuración y el conjunto de eliminación. Para obtener más información, consulte los precios de Device Farm.

En la siguiente imagen se muestra un ejemplo de la sección Dispositivos después de que una ejecución de prueba se haya parado correctamente.



Detener una ejecución (AWS CLI)

Puede ejecutar el siguiente comando para parar la ejecución de prueba especificada, donde *myARN* es el nombre de recurso de Amazon (ARN) de la ejecución de prueba.

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

Debería ver una salida similar a esta:

```
{
    "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 obtener el ARN de la ejecución, use el comando `list-runs`. El resultado debería ser similar al siguiente:

```
{
    "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 obtener información sobre el uso de Device Farm con AWS CLI, consulteReferencia de AWS CLI.

Detener una ejecución (API)

- Realice la prueba de StopRunfuncionamiento de la operación.

Para obtener más información acerca del uso de la API de Device Farm, consulte Automatización de Device Farm.

Ver una lista de ejecuciones en AWS Device Farm

Puedes usar la consola o la API de Device Farm para ver una lista de las ejecuciones de un proyecto. AWS CLI

Temas

- Visualización de una lista de ejecuciones (consola)
- Visualización de una lista de ejecuciones (AWS CLI)
- Visualización de una lista de ejecuciones (API)

Visualización de una lista de ejecuciones (consola)

1. Inicie sesión en la consola de Device Farm en https://console.aws.amazon.com/devicefarm.
2. En el panel de navegación de Device Farm, seleccione Pruebas de dispositivos móviles y, a continuación, seleccione Proyectos.
3. En la lista de proyectos, elija el proyecto que se corresponde con la lista que desea ver.

> **Tip**
>
> Puede utilizar la barra de búsqueda para filtrar la lista de proyectos por nombre.

Visualización de una lista de ejecuciones (AWS CLI)

- Ejecute el comando list-runs.

  Para ver información sobre una única ejecución, ejecute el comando get-run.

Para obtener información sobre el uso de Device Farm con AWS CLI, consulteReferencia de AWS CLI.

Visualización de una lista de ejecuciones (API)

- Llame a la API de `ListRuns`.

  Para ver información sobre una única ejecución, llame a la API `GetRun`.

Para obtener más información sobre la API de Device Farm, consulte Automatización de Device Farm.

Crear un grupo de dispositivos en AWS Device Farm

Puede usar la consola o la API de Device Farm para crear un grupo de dispositivos. AWS CLI

Temas

- Requisitos previos
- Crear un grupo de dispositivos (consola)
- Crear un grupo de dispositivos (AWS CLI)
- Crear un grupo de dispositivos (API)

Requisitos previos

- Cree una ejecución en la consola de Device Farm. Siga las instrucciones en Crear una ejecución de prueba en Device Farm. Cuando llegue a la página Seleccionar dispositivos, continúe con las instrucciones de esta sección.

Crear un grupo de dispositivos (consola)

1. En la página Seleccionar dispositivos, seleccione Crear grupo de dispositivos.
2. En Nombre, escriba un nombre que permita identificar fácilmente este grupo de dispositivos.
3. En Descripción, escriba una descripción que permita identificar fácilmente este grupo de dispositivos.

4. Si desea utilizar uno o varios criterios de selección para los dispositivos de este grupo de dispositivos, haga lo siguiente:

   a. Seleccione Crear grupo de dispositivos dinámico.

   b. Seleccione Agregar una regla.

   c. En Campo (primera lista desplegable), seleccione una de las siguientes opciones:

      - Para incluir los dispositivos por el nombre del fabricante, seleccione Fabricante del dispositivo.
      - Para incluir los dispositivos según el valor de su tipo, seleccioneFactor de forma.

   d. En Operador (segunda lista desplegable), seleccione IGUAL QUE para incluir los dispositivos cuyo valor de Campo es igual que el valor de Valor.

   e. En Valor (tercera lista desplegable), escriba o seleccione el valor que desea especificar para los valores de Campo y Operador. Si elige Plataforma para Campo, entonces las únicas selecciones disponibles serán ANDROID e IOS. De forma similar, si selecciona Factor de forma para Campo, entonces las únicas selecciones disponibles serán TELÉFONO y TABLET.

   f. Para añadir otra regla, seleccione Agregar una regla.

   g. Para eliminar una regla, elija el icono X situado junto a la regla.

   Después de crear la primera regla, en la lista de dispositivos se selecciona la casilla situada junto a cada uno de los dispositivos que coincide con la regla. Después de crear o modificar reglas, en la lista de dispositivos se selecciona la casilla situada junto a cada uno de los dispositivos que coincide con las reglas combinadas. Los dispositivos con casillas seleccionados se incluyen en el grupo de dispositivos. Los dispositivos cuyas casillas no están seleccionadas se excluyen.

5. Si desea incluir o excluir dispositivos individuales de forma manual, haga lo siguiente:

   a. Seleccione Crear grupo de dispositivos estático.

   b. Seleccione o desmarque la casilla situada junto a cada dispositivo. Puede seleccionar o borrar las casillas solo si no tiene reglas especificadas.

6. Si desea incluir o excluir todos los dispositivos que se muestran, seleccione o borre la casilla situada en la fila de encabezado de columna de la lista.

> **Important**
>
> Aunque puede utilizar las casillas de la fila de encabezados de columna para cambiar la lista de dispositivos que se muestran, eso no significa que los restantes dispositivos mostrados sean los únicos incluidos o excluidos. Para confirmar qué dispositivos se incluyen o excluyen, asegúrese de borrar el contenido de todas las casillas de la fila de encabezados de columna y, a continuación, examine las casillas.

7. Seleccione Crear.

Crear un grupo de dispositivos (AWS CLI)

- Ejecute el comando create-device-pool.

Para obtener información sobre el uso de Device Farm con AWS CLI, consulteReferencia de AWS CLI.

Crear un grupo de dispositivos (API)

- Llame a la API de `CreateDevicePool`.

Para obtener más información acerca del uso de la API de Device Farm, consulte Automatización de Device Farm.

Análisis de resultados en AWS Device Farm

En el entorno de pruebas estándar, puede utilizar la consola de Device Farm para ver informes de cada prueba de la ejecución de prueba.

Device Farm también recopila otros artefactos, como archivos, registros e imágenes, que puede descargar cuando la ejecución de prueba se ha completado.

Temas

- Trabajar con informes de pruebas en Device Farm
- Trabajar con artefactos en Device Farm

Trabajar con informes de pruebas en Device Farm

Puede utilizar la consola de Device Farm para ver los informes de las pruebas. Para obtener más información, consulte Informes en AWS Device Farm.

Temas

Requisitos previos

Configure una ejecución de prueba y compruebe que se haya completado.

1. Para crear una ejecución, consulte Crear una ejecución de prueba en Device Farm y, a continuación, vuelva a esta página.
2. Compruebe que la ejecución se haya completado. Durante la ejecución de prueba, Device Farm muestra un icono de pendiente  en la consola para las ejecuciones que están en curso. Cada dispositivo en ejecución también empezará con el icono de pendiente y, después, pasará al icono  de ejecución cuando comience la prueba. Al finalizar cada prueba, aparece un icono con el resultado de la prueba junto al nombre del dispositivo. Cuando se hayan completado todas las pruebas, el icono de pendientes situado junto a la ejecución pasará a ser el icono del resultado de la prueba. Para obtener más información, consulte Comprender los resultados de la prueba.

Comprender los resultados de la prueba

La consola de Device Farm muestra iconos que le ayudan a evaluar rápidamente el estado de la ejecución de prueba completada.

Temas

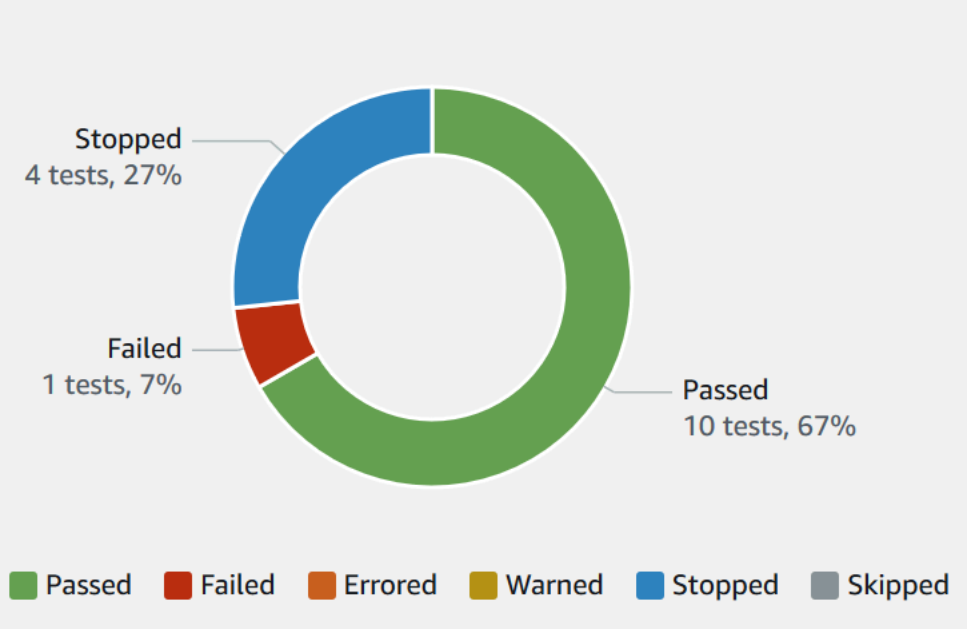
Informes de resultados de una prueba individual

Para los informes que describen una prueba individual, Device Farm muestra un icono:

Descripción	Icono
La prueba se ha completado correctamente.	
La prueba no se ha completado correctamente.	
Device Farm se saltó la prueba.	
La prueba se ha parado.	
Device Farm ha devuelto una advertencia.	
Device Farm ha devuelto un error.	

Informes de resultados de varias pruebas

Si selecciona una ejecución finalizada, Device Farm mostrará un gráfico resumido de los resultados de la prueba.



Por ejemplo, este gráfico de resultados de ejecución de prueba muestra 4 pruebas paradas, 1 prueba fallida y 10 pruebas completadas correctamente.

Los gráficos siempre están etiquetados y codificados por colores.

Visualización de informes

Puede ver los resultados de la prueba en la consola de Device Farm.

Temas

- Visualización de la página de resumen de la ejecución de prueba
- Visualización de informes de problemas únicos
- Visualización de informes de dispositivo
- Visualización de informes de conjuntos de pruebas
- Consultar los informes de pruebas
- Visualización de los datos de desempeño para un problema, dispositivo, conjunto o prueba en un informe
- Visualización de la información de registro para un problema, dispositivo, conjunto o prueba en un informe

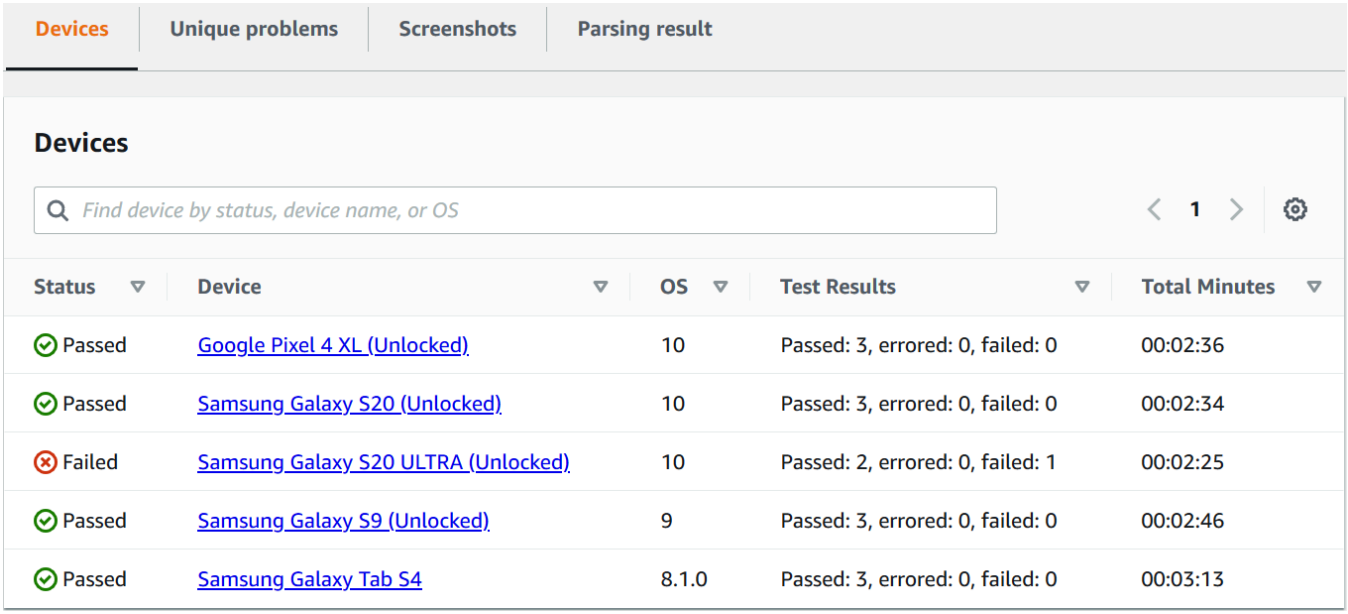
Visualización de la página de resumen de la ejecución de prueba

1. Inicie sesión en la consola de Device Farm en https://console.aws.amazon.com/devicefarm.
2. En el panel de navegación, seleccione Pruebas de dispositivos móviles y, a continuación, seleccione Proyectos.
3. En la lista de proyectos, elija el que desee incluir en la ejecución.

> **Tip**
>
> Utilice la barra de búsqueda para filtrar la lista de proyectos por nombre.

4. Elija una ejecución completada para ver su página de resumen de informe.
5. La página de resumen de la ejecución de prueba muestra información general de los resultados de las pruebas.

- La sección Problemas únicos muestra una lista con las advertencias y errores únicos. Para ver los problemas únicos, siga las instrucciones de Visualización de informes de problemas únicos.
- La sección Dispositivos muestra el número total de pruebas, según su resultado, para cada dispositivo.



  En este ejemplo, hay varios dispositivos. En la primera entrada de la tabla, el dispositivo Google Pixel 4 XL que ejecuta la versión 10 de Android informa de tres pruebas satisfactorias que tardaron 02:36 minutos en ejecutarse.

  Para ver los resultados por dispositivo, siga las instrucciones de Visualización de informes de dispositivo.
- La sección Capturas de pantalla muestra una lista de todas las capturas de pantalla que Device Farm ha capturado durante la ejecución, agrupadas por dispositivo.
- En la sección Resultado del análisis, puede descargar el resultado del análisis.

Visualización de informes de problemas únicos

1. En Problemas únicos, seleccione el problema que desee ver.
2. Elija el dispositivo. El informe muestra información sobre el problema.

   La sección Vídeo muestra una grabación en vídeo descargable de la prueba.

La sección Resultado muestra el resultado de la prueba. El estado se representa como un icono de resultado. Para obtener más información, consulte Informes de resultados de una prueba individual.

La sección Registros muestra toda la información que Device Farm ha registrado durante la prueba. Para ver esta información, siga las instrucciones de Visualización de la información de registro para un problema, dispositivo, conjunto o prueba en un informe.

La pestaña Rendimiento muestra información sobre los datos de desempeño que Device Farm ha generado durante la prueba. Para ver estos datos de desempeño, siga las instrucciones de Visualización de los datos de desempeño para un problema, dispositivo, conjunto o prueba en un informe.

La pestaña Archivos muestra una lista de todos los archivos asociados de la prueba (como archivos de registro) que puede descargar. Para descargar un archivo, elija el enlace del archivo en la lista.

La pestaña Capturas de pantalla muestra una lista de todas las capturas de pantalla que Device Farm ha capturado durante la prueba.

Visualización de informes de dispositivo

- En la sección Dispositivos, seleccione el dispositivo.

  La sección Vídeo muestra una grabación en vídeo descargable de la prueba.

  La sección Conjuntos muestra una tabla que contiene información sobre los conjuntos para el dispositivo.

  En esta tabla, la columna Resultados de las pruebas resume el número de pruebas según su resultado para cada uno de los conjuntos de pruebas que se ejecutan en el dispositivo. Estos datos también tienen un componente gráfico. Para obtener más información, consulte Informes de resultados de varias pruebas.

  Para ver los resultados por conjunto, siga las instrucciones de Visualización de informes de conjuntos de pruebas.

  La sección Registros muestra toda la información que Device Farm ha registrado para el dispositivo durante la ejecución. Para ver esta información, siga las instrucciones de

Visualización de la información de registro para un problema, dispositivo, conjunto o prueba en un informe.

La sección Rendimiento muestra información sobre los datos de desempeño que Device Farm ha generado para el dispositivo durante la ejecución. Para ver estos datos de desempeño, siga las instrucciones de Visualización de los datos de desempeño para un problema, dispositivo, conjunto o prueba en un informe.

La sección Archivos muestra una lista de conjuntos para el dispositivo y todos los archivos asociados (como archivos de registro) que puede descargar. Para descargar un archivo, elija el enlace del archivo en la lista.

La sección Capturas de pantalla muestra una lista de todas las capturas de pantalla que Device Farm ha capturado durante la ejecución para el dispositivo, agrupadas por conjunto.

Visualización de informes de conjuntos de pruebas

1. En la sección Dispositivos, seleccione el dispositivo.
2. En la sección Conjuntos, seleccione el conjunto de la tabla.

   La sección Vídeo muestra una grabación en vídeo descargable de la prueba.

   La sección Pruebas muestra una tabla que contiene información sobre las pruebas del conjunto.

   En la tabla, la columna Resultados de las pruebas muestra el resultado. Estos datos también tienen un componente gráfico. Para obtener más información, consulte Informes de resultados de varias pruebas.

   Para ver los resultados por prueba, siga las instrucciones de Consultar los informes de pruebas.

   La sección Registros muestra toda la información que Device Farm ha registrado durante la ejecución para el conjunto. Para ver esta información, siga las instrucciones de Visualización de la información de registro para un problema, dispositivo, conjunto o prueba en un informe.

   La sección Rendimiento muestra información sobre los datos de desempeño que Device Farm ha generado durante la ejecución para el conjunto. Para ver estos datos de desempeño, siga las instrucciones de Visualización de los datos de desempeño para un problema, dispositivo, conjunto o prueba en un informe.

La sección Archivos muestra una lista de pruebas para el conjunto y todos los archivos asociados (como archivos de registro) que puede descargar. Para descargar un archivo, elija el enlace del archivo en la lista.

La sección Capturas de pantalla muestra una lista de todas las capturas de pantalla que Device Farm ha capturado durante la ejecución para el conjunto, agrupadas por prueba.

Consultar los informes de pruebas

1. En la sección Dispositivos, seleccione el dispositivo.
2. En la sección Conjuntos, seleccione el conjunto.
3. En la sección Pruebas, seleccione la prueba.
4. La sección Vídeo muestra una grabación en vídeo descargable de la prueba.

   La sección Resultado muestra el resultado de la prueba. El estado se representa como un icono de resultado. Para obtener más información, consulte Informes de resultados de una prueba individual.

   La sección Registros muestra toda la información que Device Farm ha registrado durante la prueba. Para ver esta información, siga las instrucciones de Visualización de la información de registro para un problema, dispositivo, conjunto o prueba en un informe.

   La pestaña Rendimiento muestra información sobre los datos de desempeño que Device Farm ha generado durante la prueba. Para ver estos datos de desempeño, siga las instrucciones de Visualización de los datos de desempeño para un problema, dispositivo, conjunto o prueba en un informe.

   La pestaña Archivos muestra una lista de todos los archivos asociados de la prueba (como archivos de registro) que puede descargar. Para descargar un archivo, elija el enlace del archivo en la lista.

   La pestaña Capturas de pantalla muestra una lista de todas las capturas de pantalla que Device Farm ha capturado durante la prueba.

Visualización de los datos de desempeño para un problema, dispositivo, conjunto o prueba en un informe

> **Note**
>
> Device Farm recopila datos de rendimiento de los dispositivos solo para dispositivos Android en este momento.

En la pestaña Rendimiento se muestra la siguiente información:

- El gráfico CPU muestra el porcentaje de CPU que usa la aplicación en un solo núcleo durante el problema, dispositivo, conjunto o prueba seleccionados (en el eje vertical) a lo largo del tiempo (en el eje horizontal).

  El eje vertical se expresa en porcentajes, desde 0 % hasta el porcentaje máximo registrado.

  Este porcentaje puede ser superior al 100 % si la aplicación utiliza más de un núcleo. Por ejemplo, si tres núcleos están al 60 % de su uso, el porcentaje que se muestra es el 180 %.
- El gráfico Memoria muestra el número de MB que usa la aplicación durante el problema, dispositivo, conjunto o prueba seleccionados (en el eje vertical) a lo largo del tiempo (en el eje horizontal).

  El eje vertical se expresa en MB, desde 0 MB hasta el número máximo de MB registrados.
- El gráfico Subprocesos muestra el número de subprocesos usados durante el problema, dispositivo, conjunto o prueba seleccionados (en el eje vertical) a lo largo del tiempo (en el eje horizontal).

  El eje vertical se expresa en número de subprocesos, desde cero subprocesos hasta el número máximo de subprocesos registrados.

En todos los casos, el eje horizontal está representado en segundos, desde el inicio y el final de la ejecución para el problema, dispositivo, conjunto o prueba seleccionados.

Para mostrar información para un punto de datos específico, ponga en pausa el gráfico deseado en el segundo deseado a lo largo del eje horizontal.

Visualización de la información de registro para un problema, dispositivo, conjunto o prueba en un informe

En la sección de Registros, se muestra lo siguiente:

- Fuente representa la fuente de una entrada de registro. Los valores posibles son:
  - Herramienta representa una entrada de registro creada por Device Farm. Estas entradas de log suelen crearse durante los eventos de comienzo y parada.
  - Dispositivo representa una entrada de registro creada por el dispositivo. Para Android, estas entradas de log son compatibles con logcat. Para iOS, estas entradas de registro son compatibles con syslog.
  - Prueba representa una entrada de registro creada por una prueba o su marco de pruebas.
- Tiempo representa el tiempo transcurrido entre la primera entrada de registro y esta entrada de registro. El tiempo se expresa en formato *`MM:SS.SSS`*, donde *`M`* representa los minutos y *`S`* representa los segundos.
- PID representa el identificador de proceso (PID) que creó la entrada de registro. Todas las entradas de registro creadas por una aplicación en un dispositivo tienen el mismo PID.
- Nivel representa el nivel de registro para la entrada de registro. Por ejemplo, para `Logger.debug("This is a message!")`, el valor de Nivel que se registra es `Debug`. Estos son los valores posibles:
  - Alerta
  - Critical
  - Debug
  - Emergencia
  - Error
  - Con errores
  - Con error
  - Información
  - Interno
  - Aviso
  - Passed
  - Skipped
  - Stopped

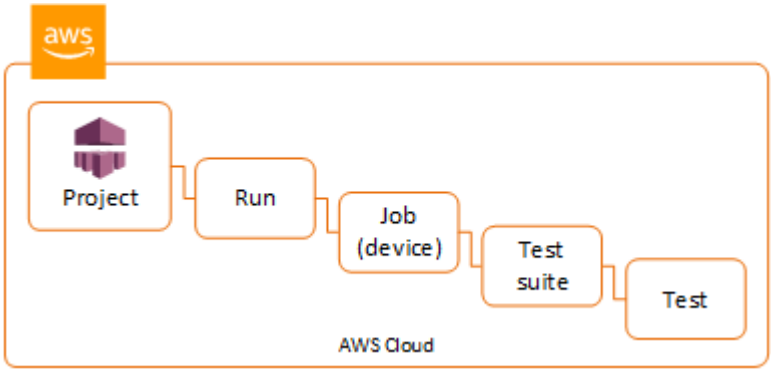
  - Detallado
  - Con advertencia
  - Advertencia
- Etiqueta representa los metadatos arbitrarios para la entrada de registro. Por ejemplo, logcat de Android puede usar esto para describir qué parte del sistema creó la entrada de log (por ejemplo, `ActivityManager`).
- Mensaje representa el mensaje o los datos para la entrada de registro. Por ejemplo, para `Logger.debug("Hello, World!")`, el valor de Mensaje que se registra es `"Hello, World!"`.

Para mostrar solo una parte de la información:

- Para mostrar todas las entradas de registro que coinciden con un valor para una columna específica, escriba el valor en la barra de búsqueda. Por ejemplo, para mostrar todas las entradas de registro cuyo valor de Fuente es `Harness`, escriba **`Harness`** en la barra de búsqueda.
- Para quitar todos los caracteres de un cuadro de encabezado de columna, elija la X en ese cuadro de encabezado de la columna. Eliminar todos los caracteres de un cuadro de encabezado de columna es lo mismo que escribir ***** en ese cuadro de encabezado de columna.

Para descargar toda la información de registro del dispositivo, incluyendo todos los conjuntos y las pruebas que se han ejecutado, seleccione Descargar registros.

Trabajar con artefactos en Device Farm



Device Farm recopila artefactos como informes, archivos de registro e imágenes, de cada prueba de la ejecución.

Puede descargar los artefactos creados durante la ejecución de prueba:

Archivos

Archivos generados durante la ejecución de prueba, como los informes de Device Farm. Para obtener más información, consulte Trabajar con informes de pruebas en Device Farm.

Registros

Salida de cada prueba de la ejecución de prueba.

Capturas de pantalla

Imágenes de las pantallas registradas para cada prueba de la ejecución de prueba.

Uso de artefactos (consola)

1. En la página de la ejecución de prueba, seleccione un dispositivo móvil en Dispositivos.
2. Para descargar un archivo, selecciónelo en Archivos.
3. Para descargar los registros de la ejecución de prueba, en Registros, seleccione Descargar registros.
4. Para descargar una captura de pantalla, seleccione una captura de pantalla de Capturas de pantalla.

Para obtener más información acerca de cómo descargar artefactos en un entorno de pruebas personalizado, consulte Uso de artefactos en un entorno de pruebas personalizado.

Uso de artefactos (AWS CLI)

Puede utilizarla AWS CLI para enumerar los artefactos de las pruebas realizadas.

Temas

- Paso 1: Obtener los nombres de recurso de Amazon (ARN)
- Paso 2: Crear una lista con los artefactos
- Paso 3: Descargar los artefactos

Paso 1: Obtener los nombres de recurso de Amazon (ARN)

Los artefactos se pueden enumerar por ejecución, trabajo, conjunto de pruebas o prueba. Necesita el ARN correspondiente. En esta tabla se muestra el ARN de entrada para cada uno de los comandos de la AWS CLI lista:

AWS CLI Comando de lista	ARN requerido
list-projects	Este comando devuelve todos los proyectos y no requiere un ARN.
list-runs	`project`
list-jobs	`run`
list-suites	`job`
list-tests	`suite`

Por ejemplo, para encontrar un ARN de prueba, ejecute list-tests con el ARN del conjunto de pruebas como parámetro de entrada.

Ejemplo:

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

La respuesta incluye un ARN de prueba para cada prueba del conjunto de pruebas.

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

Paso 2: Crear una lista con los artefactos

El comando AWS CLI list-artifacts devuelve una lista de artefactos, como archivos, capturas de pantalla y registros. Cada artefacto tiene una URL que le permite descargar el archivo.

- Llame a list-artifacts especificando el ARN de una ejecución, un trabajo, un conjunto de pruebas o una prueba. Especifique un tipo de archivo, registro o captura de pantalla.

  En este ejemplo, se devuelve una URL de descarga para cada artefacto disponible de una prueba individual:

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

  La respuesta contiene una URL de descarga para cada artefacto.

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

Paso 3: Descargar los artefactos

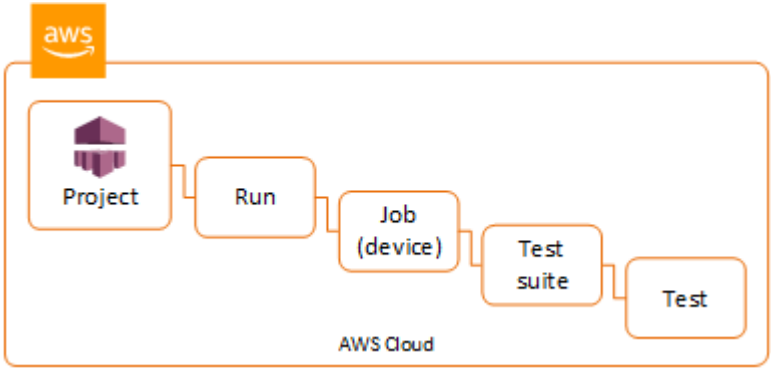
- Descargue el artefacto mediante la dirección URL del paso anterior. En este ejemplo se utiliza curl para descargar un archivo de salida de Android Appium Java:

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

Uso de artefactos (API)

El ListArtifactsmétodo de la API Device Farm devuelve una lista de artefactos, como archivos, capturas de pantalla y registros. Cada artefacto tiene una URL que le permite descargar el archivo.

Uso de artefactos en un entorno de pruebas personalizado



En un entorno de pruebas personalizado, Device Farm recopila artefactos como informes personalizados, archivos de registro e imágenes. Estos artefactos de prueba están disponibles para cada dispositivo de la ejecución de prueba.

Puede descargar estos artefactos creados durante la ejecución de prueba:

Salida de la especificación de prueba

La salida de la ejecución de los comandos en el archivo YAML de la especificación de prueba.

Artefactos de clientes

Archivo comprimido que contiene los artefactos de la ejecución de prueba. Se configura en la sección artifacts (artefactos) del archivo YAML de la especificación de prueba.

Script del shell de la especificación de prueba

Archivo de script del shell intermedio creado a partir del archivo YAML. Dado que se utiliza en la ejecución de prueba, el script del shell se puede usar para depurar el archivo YAML.

Archivo de la especificación de prueba

Archivo YAML utilizado en la ejecución de prueba.

Para obtener más información, consulte Trabajar con artefactos en Device Farm.

Etiquetado de recursos en AWS Device Farm

AWS Device Farm funciona con la API de etiquetado de grupos de recursos de AWS. Esta API le permite administrar los recursos de su cuenta de AWS con etiquetas. Puede agregar etiquetas a recursos, como proyectos y ejecuciones de prueba.

Puede usar etiquetas para:

- Organizar su factura de AWS de manera que refleje su propia estructura de costos. Para ello, regístrese para obtener una factura de su cuenta de AWS que incluya valores de clave de etiquetas. A continuación, para ver los costos de los recursos combinados, organice la información de facturación de acuerdo con los recursos con los mismos valores de clave de etiquetas. Por ejemplo, puede etiquetar varios recursos con un nombre de aplicación y luego organizar su información de facturación para ver el costo total de la aplicación en distintos servicios. Para obtener más información, consulte Asignación y etiquetado de costos en Acerca de la administración de facturación y costos de AWS.
- Controlar el acceso a través de políticas de IAM. Para ello, cree una política que permita el acceso a un recurso o conjunto de recursos mediante una condición de valor de etiqueta.
- Identificar y administrar ejecuciones que tienen ciertas propiedades como etiquetas, como la ramificación que se usó para las pruebas.

Para obtener más información sobre el etiquetado de recursos, consulte el documento técnico Prácticas recomendadas de etiquetado.

Temas

- Etiquetado de recursos
- Buscar recursos por etiquetas
- Eliminación de etiquetas de recursos

Etiquetado de recursos

La API de etiquetado de grupos de recursos de AWS le permite agregar, quitar o modificar etiquetas en los recursos. Para obtener más información, consulte la referencia de API de etiquetado de grupos de recursos de AWS.

Para etiquetar un recurso, utilice la operación `TagResources` operación desde el punto de enlace de `resourcegroupstaggingapi`. Esta operación usa una lista de ARN de los servicios compatibles y una lista de pares clave-valor. El valor es opcional. Una cadena vacía indica que no debe haber ningún valor para esa etiqueta. Por ejemplo, el siguiente ejemplo de Python etiqueta una serie de ARN de proyecto con la etiqueta `build-config` con el 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"})
```

Un valor de etiqueta no es obligatorio. Para establecer una etiqueta sin valor, utilice una cadena vacía ("") al especificar un valor. Una etiqueta solo puede tener un valor. Cualquier valor anterior que una etiqueta tenga para un recurso se sobrescribirá con el nuevo valor.

Buscar recursos por etiquetas

Para buscar recursos por sus etiquetas, utilice la operación `GetResources` desde el punto de enlace `resourcegrouptaggingapi`. Esta operación usa una serie de filtros, ninguno de los cuales es necesario, y devuelve los recursos que coinciden con los criterios dados. Sin filtros, se devuelven todos los recursos etiquetados. La operación `GetResources` le permite filtrar recursos basados en

- Valor de etiqueta
- Tipo de recurso (por ejemplo, `devicefarm:run`)

Para obtener más información, consulte la referencia de API de etiquetado de grupos de recursos de AWS.

En el siguiente ejemplo se buscan sesiones de prueba del explorador de de escritorio en Device Farm (recursos `devicefarm:testgrid-session`) con la etiqueta `stack` que tiene el valor `production`:

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

Eliminación de etiquetas de recursos

Para quitar una etiqueta, utilice la operación `UntagResources`, especificando una lista de recursos y las etiquetas que quiere quitar:

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

Trabajar con tipos de pruebas en AWS Device Farm

En esta sección se describe el soporte de Device Farm para marcos de prueba, así como los tipos de prueba integradas.

Marcos de pruebas

Device Farm admite estos marcos de automatización de pruebas móviles:

Marcos de pruebas de aplicaciones Android

- Trabajar con Appium y AWS Device Farm
- Trabajar con instrumentación para Android y AWS Device Farm

Marcos de pruebas de aplicaciones iOS

- Trabajar con Appium y AWS Device Farm
- Uso de XCTest para iOS y AWS Device Farm
- XCTest UI

Marcos de pruebas de aplicaciones web

Las aplicaciones web son compatibles con Appium. Para obtener más información sobre cómo llevar sus pruebas a Appium, consulte Trabajar con Appium y AWS Device Farm.

Marcos en un entorno de prueba personalizado

Device Farm no admite la personalización del entorno de pruebas para el marco XCTests. Para obtener más información, consulte Trabajo con entornos de pruebas personalizados.

Compatibilidad con versiones de Appium

Para las pruebas que se ejecutan en un entorno personalizado, Device Farm admite Appium versión 1. Para obtener más información, consulte Entornos de prueba.

Tipos de pruebas integradas

Las pruebas integradas permiten probar la aplicación en varios dispositivos sin tener que escribir ni mantener scripts de automatización de pruebas. Device Farm ofrece un tipo de prueba integrada:

- Integrado: fuzzing (Android e iOS)

Trabajar con Appium y AWS Device Farm

En esta sección se describe cómo configurar, empaquetar y cargar pruebas de Appium en Device Farm. Appium es una herramienta de código abierto que permite automatizar aplicaciones web nativas para dispositivos móviles. Para obtener más información, consulte Introduction to Appium en el sitio web de Appium.

Para ver una aplicación de muestra y enlaces a pruebas de funcionamiento, consulte Device Farm Sample App para Android y Device Farm Sample App para iOS en adelante GitHub.

Compatibilidad de versiones

La compatibilidad con varios marcos y lenguajes de programación depende del lenguaje utilizado.

Device Farm es compatible con todas las versiones de servidor Appium 1.x y 2.x. En el caso de Android, puedes elegir cualquier versión principal de Appium con `devicefarm-cli`. Por ejemplo, para utilizar la versión 2 del servidor Appium, añada estos comandos a su archivo YAML de especificación de prueba:

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

Para iOS, puede elegir versiones específicas de Appium con los comandos `avm` o`npm`. Por ejemplo, para utilizar el comando `avm` para configurar la versión del servidor Appium 2.1.2, añada estos comandos a su archivo YAML de especificación de prueba:

```
phases:
```

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

Con el comando npm para usar la última versión de Appium 2, añada estos comandos a su archivo YAML con especificaciones de prueba:

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

Para obtener más información sobre `devicefarm-cli` o los comandos de la CLI, consulte la Referencia de comandos de la CLI.

Para usar todas las características del marco, como las anotaciones, seleccione un entorno de prueba personalizado y use la AWS CLI o la consola `Device Farm` para cargar una especificación de prueba personalizada.

Temas

- Configurar el paquete de prueba de Appium
- Crear un archivo de paquete comprimido
- Cargar el paquete de prueba en Device Farm
- Realizar capturas de pantalla de sus pruebas (opcional)

Configurar el paquete de prueba de Appium

Utilice las siguientes instrucciones para configurar el paquete de pruebas.

Java (JUnit)

1. Modifique `pom.xml` para establecer el empaquetamiento como un archivo 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` para compilar las pruebas en un archivo JAR.

   El siguiente complemento compila el código fuente de la prueba (todo lo que haya en el directorio `src/test`) en un archivo 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` para compilar dependencias como archivos JAR.

   El siguiente complemento copiará sus dependencias en el directorio `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. Guarde el siguiente ensamblaje XML en `src/main/assembly/zip.xml`.

   El siguiente XML es una definición de ensamblaje que, cuando está configurado, indica a Maven que compile un archivo .zip que contenga todo lo que haya en la raíz del directorio de salida de compilación y en el directorio `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` para empaquetar las pruebas y todas las dependencias en un único archivo .zip.

   El siguiente complemento utiliza el ensamblaje anterior para crear un archivo .zip denominado `zip-with-dependencies` en el directorio de salida de compilación cada vez que se ejecuta mvn package:

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

> **Note**
>
> Si recibe un error que le informa de que la anotación no se admite en 1.3, añada lo siguiente a `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 establecer el empaquetamiento como un archivo 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` para compilar las pruebas en un archivo JAR.

   El siguiente complemento compila el código fuente de la prueba (todo lo que haya en el directorio `src/test`) en un archivo 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` para compilar dependencias como archivos JAR.

   El siguiente complemento copiará sus dependencias en el directorio `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. Guarde el siguiente ensamblaje XML en `src/main/assembly/zip.xml`.

   El siguiente XML es una definición de ensamblaje que, cuando está configurado, indica a Maven que compile un archivo .zip que contenga todo lo que haya en la raíz del directorio de salida de compilación y en el directorio `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` para empaquetar las pruebas y todas las dependencias en un único archivo .zip.

El siguiente complemento utiliza el ensamblaje anterior para crear un archivo .zip denominado `zip-with-dependencies` en el directorio de salida de compilación cada vez que se ejecuta mvn package:

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

Note

Si recibe un error que le informa de que la anotación no se admite en 1.3, añada lo siguiente a `pom.xml`:

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

Node.JS

Para empaquetar sus pruebas de Appium Node.js y cargarlas en Device Farm, debe instalar lo siguiente en su equipo local:

- Node Version Manager (nvm)

  Utilice esta herramienta cuando desarrolle y empaquete sus pruebas de forma que no se incluyan dependencias innecesarias en el paquete de pruebas.
- Node.js
- npm-bundle (instalado globalmente)

1. Verifique que nvm esté presente.

   ```
   command -v nvm
   ```

   Debería ver `nvm` como salida.

   Para obtener más información, consulte nvm on GitHub.
2. Ejecute este comando para instalar Node.js:

   ```
   nvm install node
   ```

   Puede especificar una versión concreta de Node.js:

   ```
   nvm install 11.4.0
   ```
3. Verifique que esté en uso la versión correcta de Node:

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

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

Python

1. Le recomendamos que configure el módulo virtualenv de Python para desarrollar y empaquetar pruebas de forma que no se incluyan dependencias innecesarias en el paquete de la aplicación.

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

> **Tip**
>
> - No cree un módulo virtualenv de Python con la opción `--system-site-packages`, ya que hereda paquetes del directorio site-packages global. Esto puede dar lugar a que se incluyan en el entorno virtual dependencias que las pruebas no necesiten.
> - También debería comprobar que las pruebas no utilizan dependencias que dependan de bibliotecas nativas, ya que esas bibliotecas nativas podrían no estar en la instancia en la que se ejecuten estas pruebas.

2. Instale py.test en el entorno virtual.

```
$ pip install pytest
```

3. Instale el cliente de Appium Python en su entorno virtual.

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

4. A menos que especifique una ruta diferente en modo personalizado, Device Farm espera que sus pruebas se almacenen en `tests/`. Puede usar `find` para mostrar todos los archivos dentro de una carpeta:

```
$ find tests/
```

Confirme que estos archivos contienen conjuntos de pruebas que quiere ejecutar en Device Farm

```
tests/
```

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

5. Ejecute este comando desde la carpeta de área de trabajo del entorno virtual para mostrar una lista de las pruebas sin ejecutarlas.

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

   Confirme que la salida muestra las pruebas que desea ejecutar en Device Farm.

6. Limpie todos los archivos almacenados en caché en sus pruebas/ carpeta:

```
$ 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. Ejecute el comando siguiente en su espacio de trabajo para generar el archivo requirements.txt:

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

Ruby

Para empaquetar sus pruebas de Appium Ruby y cargarlas en Device Farm, debe instalar lo siguiente en su equipo local:

- Ruby Version Manager (RVM)

  Utilice esta herramienta de línea de comandos cuando desarrolle y empaquete sus pruebas de forma que no se incluyan dependencias innecesarias en el paquete de pruebas.
- Ruby
- Bundler (Esta gema normalmente se instala con Ruby).

1. Instale las claves requeridas, RVM y Ruby. Para obtener instrucciones, consulte Instalación de RVM en el sitio web de RVM.

   Una vez realizada la instalación, vuelva a cargar el terminal cerrando la sesión y volviéndola a iniciar a continuación.

> **Note**
>
> RVM se carga como función solo para el shell de bash.

2. Verifique que rvm está instalado correctamente.

```
command -v rvm
```

Debería ver `rvm` como salida.

3. Si desea instalar una versión específica de Ruby (por ejemplo, la *2.5.3*) ejecute el siguiente comando:

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

Verifique que está en la versión solicitada de Ruby:

```
ruby -v
```

4. Configure el paquete para compilar paquetes para las plataformas de prueba que desee:

```
bundle config specific_platform true
```

5. Actualice su archivo .lock para añadir las plataformas necesarias para ejecutar las pruebas.

   - Si está compilando pruebas para ejecutarlas en dispositivos Android, ejecute este comando para configurar el Gemfile de manera que use dependencias para el host de pruebas de Android:

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

   - Si está compilando pruebas para ejecutarlas en dispositivos iOS, ejecute este comando para configurar el Gemfile de manera que use dependencias para el host de pruebas de iOS:

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

6. Por lo general, la gema bundler está instalada de forma predeterminada. Si no es así, instálela:

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

Crear un archivo de paquete comprimido

> **Warning**
>
> En Device Farm, la estructura de carpetas de los archivos del paquete de prueba comprimido es importante, y algunas herramientas de archivado cambiarán la estructura del archivo ZIP de forma implícita. Le recomendamos que utilice las utilidades de línea de comandos que se especifican a continuación en lugar de utilizar las utilidades de archivado integradas en el administrador de archivos del escritorio local (como Finder o el Explorador de Windows).

Ahora, agrupe las pruebas para Device Farm.

Java (JUnit)

Cree y empaquete las pruebas:

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

El archivo `zip-with-dependencies.zip` se creará como resultado. Este es el paquete de prueba.

Java (TestNG)

Cree y empaquete las pruebas:

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

El archivo `zip-with-dependencies.zip` se creará como resultado. Este es el paquete de prueba.

Node.JS

1. Revise su proyecto.

   Asegúrese de que se encuentra en el directorio raíz del proyecto. Puede ver `package.json` en el directorio raíz.

2. Ejecute este comando para instalar sus dependencias locales.

```
npm install
```

Este comando también crea una carpeta `node_modules` en el directorio actual.

> **Note**
>
> En este momento, debería poder ejecutar las pruebas localmente.

3. Ejecute este comando para empaquetar los archivos en su carpeta actual en un archivo *.tgz. El archivo se nombra utilizando la propiedad `name` en su archivo `package.json`.

```
npm-bundle
```

Este archivo tar (.tgz) contiene todo el código y todas las dependencias.

4. Ejecute este comando para agrupar el archivo tar (archivo *.tgz) generado en el paso anterior en un archivo comprimido:

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

Este es el archivo `MyTests.zip` que carga en Device Farm en el procedimiento siguiente.

Python

Python 2

Genere un archivo de los paquetes requeridos con Python (llamado «wheelhouse») usando pip:

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

Empaquete su wheelhouse, pruebas y requisitos de pip en un archivo zip para Device Farm:

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

Python 3

Empaquete sus pruebas y requisitos de pip en un archivo zip:

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

Ruby

1. Ejecute este comando para crear un entorno virtual de Ruby:

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

2. Ejecute este comando para utilizar el entorno que acaba de crear:

```
rvm gemset use myGemset
```

3. Revise el código fuente.

   Asegúrese de que se encuentra en el directorio raíz del proyecto. Puede ver `Gemfile` en el directorio raíz.

4. Ejecute este comando para instalar sus dependencias locales y todas las gemas del `Gemfile`:

```
bundle install
```

> **Note**
>
> En este momento, debería poder ejecutar las pruebas localmente. Utilice este comando para ejecutar una prueba localmente:
>
> ```
> bundle exec $test_command
> ```

5. Empaquete las gemas en la carpeta `vendor/cache`.

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

6. Ejecute el comando siguiente para empaquetar el código fuente, junto con todas sus dependencias, en un solo archivo comprimido:

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

Este es el archivo `MyTests.zip` que carga en Device Farm en el procedimiento siguiente.

Cargar el paquete de prueba en Device Farm

Puede utilizar la consola de Device Farm para cargar las pruebas.

1. Inicie sesión en la consola de Device Farm en https://console.aws.amazon.com/devicefarm.
2. En el panel de navegación de Device Farm, seleccione Pruebas de dispositivos móviles y, a continuación, seleccione Proyectos.
3. Si es un usuario nuevo, seleccione Nuevo proyecto, introduzca un nombre para el proyecto y, a continuación, seleccione Enviar.

   Si ya dispone de un proyecto, puede seleccionarlo para cargar las pruebas en él.
4. Abra el proyecto y, a continuación, seleccione Crear una nueva ejecución.
5. Para pruebas nativas de iOS y Android

   En la página Elegir aplicación, seleccione Aplicación móvil y Elegir archivo para cargar el paquete distribuible de su aplicación.

   > **Note**
   >
   > El archivo debe ser un `.apk` de Android o un `.ipa` de iOS. Las aplicaciones de iOS se deben compilar para dispositivos reales, no para el simulador.

   Para pruebas de aplicaciones web móviles

   En la página Elegir aplicación, seleccione Aplicación web.
6. Asigne a su prueba un nombre apropiado. Puede contener cualquier combinación de espacios o signos de puntuación.
7. Seleccione Siguiente.

8. En la página Configurar, en la sección Configurar marco de pruebas, seleccione el ***lenguaje*** Appium y, a continuación, Elegir archivo.
9. Busque y elija el archivo .zip que contiene las pruebas. El archivo .zip debe respetar el formato que se describe en Configurar el paquete de prueba de Appium.
10. Seleccione Ejecutar la prueba en un entorno personalizado. Este entorno de ejecución permite un control total sobre la configuración de pruebas, desglose e invocación, así como elegir versiones específicas de los tiempos de ejecución y el servidor Appium. Puede configurar su entorno personalizado a través del archivo de especificaciones de prueba. Para obtener más información, consulte Trabajo con entornos de pruebas personalizados en AWS Device Farm.
11. Seleccione Siguiente paso y, a continuación, siga las instrucciones para seleccionar dispositivos e inicie la ejecución. Para obtener más información, consulte Crear una ejecución de prueba en Device Farm.

> **Note**
>
> Device Farm no modifica las pruebas de Appium.

Realizar capturas de pantalla de sus pruebas (opcional)

Puede realizar capturas de pantalla como parte de las pruebas.

Device Farm establece la propiedad `DEVICEFARM_SCREENSHOT_PATH` en una ruta completa del sistema de archivos local donde Device Farm espera que se almacenen las capturas de pantalla de Appium. El directorio específico de la prueba donde se almacenan las capturas de pantalla se define en tiempo de ejecución. Las capturas de pantalla se extraen en los informes de Device Farm automáticamente. Para ver las capturas de pantalla en la consola de Device Farm, seleccione la sección Capturas de pantalla.

Para obtener más información sobre cómo realizar capturas de pantalla en pruebas de Appium, consulte Realizar una captura de pantalla en la documentación de la API de Appium.

Trabajar con pruebas de Android en AWS Device Farm

Device Farm es compatible con varios tipos de pruebas de automatización para dispositivos Android y dos pruebas integradas.

Marcos de pruebas de aplicaciones Android

Las siguientes pruebas están disponibles para dispositivos Android.

- Trabajar con Appium y AWS Device Farm
- Trabajar con instrumentación para Android y AWS Device Farm

Tipos de pruebas integradas para Android

Existe un tipo de prueba integrada disponible para dispositivos Android.

- Integrado: fuzzing (Android e iOS)

Trabajar con instrumentación para Android y AWS Device Farm

Device Farm ofrece soporte para instrumentación (JUnit, Espresso, Robotium o cualquier otra prueba basada en instrumentación) para Android.

Device Farm también ofrece una aplicación Android de ejemplo y enlaces a pruebas activas en tres marcos de automatización de Android, incluida la instrumentación (Espresso). La aplicación de muestra Device Farm para Android está disponible para su descarga en GitHub.

Temas

- ¿Qué es la instrumentación?
- Carga de las pruebas de instrumentación para Android
- Realizar capturas de pantalla en pruebas de instrumentación para Android
- Consideraciones adicionales para pruebas de instrumentación para Android
- Análisis de prueba en modo estándar

¿Qué es la instrumentación?

La instrumentación para Android le permite invocar métodos de devolución de llamada en el código de la prueba, de manera que pueda seguir paso a paso todo el ciclo de vida de un componente como si lo estuviera depurando. Para obtener más información, consulte Instrumented tests (Pruebas instrumentalizadas) en la sección Aspectos básicos de las pruebas de la documentación Herramientas para el desarrollador de Android.

Carga de las pruebas de instrumentación para Android

Utilice la consola de Device Farm para cargar las pruebas.

1. Inicie sesión en la consola de Device Farm en https://console.aws.amazon.com/devicefarm.
2. En el panel de navegación de Device Farm, seleccione Pruebas de dispositivos móviles y, a continuación, seleccione Proyectos.
3. En la lista de proyectos, seleccione el proyecto en el que desea cargar las pruebas.

   > **Tip**
   >
   > Puede utilizar la barra de búsqueda para filtrar la lista de proyectos por nombre. Para crear un proyecto, siga las instrucciones de Crear un proyecto en AWS Device Farm.

4. Si se muestra el botón Crear una nueva ejecución, selecciónelo.
5. En la página Elegir aplicación, seleccione Elegir archivo.
6. Busque y elija el archivo de aplicación de Android. El archivo debe ser un archivo .apk.
7. Seleccione Siguiente.
8. En la página Configurar, en la sección Configurar marco de pruebas, seleccione Instrumentación y, a continuación, seleccione Elegir archivo.
9. Busque y elija el archivo .apk que contiene las pruebas.
10. Seleccione Siguiente y, a continuación, complete el resto de instrucciones para seleccionar dispositivos e inicie la ejecución.

Realizar capturas de pantalla en pruebas de instrumentación para Android

Puede realizar capturas de pantalla como parte de las pruebas de instrumentación para Android.

Para realizar capturas de pantalla, llame a uno de los siguientes métodos:

- Para Robotium, llame al método `takeScreenShot` (por ejemplo, `solo.takeScreenShot();`).
- Para Spoon, llame al método `screenshot`, por ejemplo:

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

Durante una ejecución de prueba, Device Farm obtiene automáticamente capturas de pantalla de las siguientes ubicaciones de los dispositivos, si las hay. A continuación, las añade a los informes de las pruebas:

- `/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`

Consideraciones adicionales para pruebas de instrumentación para Android

Animaciones del sistema

Según la documentación de Android para pruebas de Espresso, se recomienda que las animaciones del sistema estén desactivadas al realizar pruebas en dispositivos reales. Device Farm deshabilita automáticamente los ajustes Window Animation Scale, Transition Animation Scale y Animator Duration Scale cuando se ejecuta con el ejecutor de pruebas de instrumentación android.support.test.Runner.AndroidJ. UnitRunner

Grabadores de pruebas

Device Farm es compatible con marcos, como Robotium, que tienen herramientas de record-and-playback creación de scripts.

Análisis de prueba en modo estándar

En el modo estándar de ejecución, Device Farm analiza el conjunto de pruebas e identifica las clases y métodos de prueba únicos que se ejecutarán. Esto se hace a través de una herramienta llamada Dex Test Parser.

Cuando se introduce un archivo.apk de instrumentación de Android como entrada, el analizador devuelve los nombres de los métodos completos de las pruebas que coinciden con las convenciones JUnit 3 y JUnit 4.

Para probar esto en un entorno local:

1. Descargue el documento binario `dex-test-parser`.

2. Ejecute el siguiente comando para obtener la lista de métodos de prueba que se ejecutarán en Device Farm:

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

Uso de pruebas de iOS en AWS Device Farm

Device Farm es compatible con varios tipos de pruebas de automatización para dispositivos iOS y una prueba integrada.

Marcos de pruebas de aplicaciones iOS

Las siguientes pruebas están disponibles para dispositivos iOS.

- Trabajar con Appium y AWS Device Farm
- Uso de XCTest para iOS y AWS Device Farm
- XCTest UI

Tipos de pruebas integradas para iOS

Actualmente hay un tipo de prueba integrada disponible para dispositivos iOS.

- Integrado: fuzzing (Android e iOS)

Uso de XCTest para iOS y AWS Device Farm

Con Device Farm, puede utilizar el marco XCTest para probar la aplicación en dispositivos reales. Para obtener más información acerca de XCTest, consulte Testing Basics en la sección Testing with Xcode.

Para ejecutar una prueba, debe crear los paquetes de la ejecución de prueba y cargar estos paquetes en Device Farm.

Temas

- Creación de los paquetes para su ejecución con XCTest
- Carga de los paquetes para su ejecución con XCTest en Device Farm

Creación de los paquetes para su ejecución con XCTest

Para probar la aplicación mediante el marco XCTest, Device Farm requiere lo siguiente:

- El paquete de la aplicación como un archivo `.ipa`.
- El paquete de XCTest como un archivo `.zip`.

Para crear estos paquetes, utilice la salida de la compilación que Xcode genera. Siga los pasos que se describen a continuación para crear los paquetes de modo que pueda cargarlos en Device Farm.

Para generar la salida de la compilación para su aplicación

1. Abra el proyecto de la aplicación en Xcode.
2. En el menú desplegable de esquema en la barra de herramientas de Xcode, seleccione Dispositivo iOS genérico como destino.
3. En el menú Producto, seleccione Compilar para y, a continuación, seleccione Pruebas.

Para crear el paquete de la aplicación

1. En el navegador del proyecto Xcode, en Productos, abra el menú contextual del archivo denominado *`app-project-name`*`.app`. A continuación, seleccione Mostrar en Finder. Finder abre una carpeta con el nombre `Debug-iphoneos`, que contiene la salida que Xcode generó para su compilación de prueba. Esta carpeta incluye su archivo `.app`.
2. En Finder, cree una nueva carpeta y asígnele el nombre `Payload`.
3. Copie el archivo *`app-project-name`*`.app` y péguelo en la carpeta `Payload`.
4. Abra el menú contextual de la carpeta `Payload` y seleccione Comprimir "Payload". Se crea un archivo denominado `Payload.zip`.
5. Cambie el nombre y la extensión del archivo `Payload.zip` a *`app-project-name`*`.ipa`.

   En un paso posterior, proporcionará este nombre de archivo a Device Farm. Para que sea más fácil encontrar el archivo, es recomendable que lo mueva a otra ubicación, como el escritorio.
6. Si lo prefiere, puede eliminar la carpeta `Payload` y el archivo `.app` que contiene.

Para crear el paquete de XCTest

1. En Finder, en el directorio `Debug-iphoneos`, abra el menú contextual del archivo *app-project-name*`.app`. A continuación, seleccione Mostrar contenidos del paquete.
2. En el contenido del paquete, abra la carpeta `Plugins`. Esta carpeta contiene un archivo denominado *app-project-name*`.xctest`.
3. Abra el menú contextual de este archivo y seleccione Comprimir "***app-project-name*`.xctest`**". Se crea un archivo denominado *app-project-name*`.xctest.zip`.

   En un paso posterior, proporcionará este nombre de archivo a Device Farm. Para que sea más fácil encontrar el archivo, es recomendable que lo mueva a otra ubicación, como el escritorio.

Carga de los paquetes para su ejecución con XCTest en Device Farm

Utilice la consola de Device Farm para cargar los paquetes de la prueba.

1. Inicie sesión en la consola de Device Farm en https://console.aws.amazon.com/devicefarm.
2. Si todavía no tiene un proyecto, cree uno. Para conocer los pasos necesarios para crear un proyecto, consulte Crear un proyecto en AWS Device Farm.

   De lo contrario, en el panel de navegación de Device Farm, seleccione Pruebas de dispositivos móviles y, a continuación, seleccione Proyectos.
3. Seleccione el proyecto que desea utilizar para ejecutar la prueba.
4. Seleccione Crear una nueva ejecución.
5. En la página Elegir aplicación, seleccione Aplicación móvil.
6. Seleccione Elegir archivo.
7. Desplácese hasta el archivo `.ipa` de la aplicación y cárguelo.

   > **Note**
   >
   > El paquete `.ipa` debe estar compilado para pruebas.

8. Una vez que finalice el proceso de carga, seleccione Siguiente.
9. En la página Configurar, en la sección Configurar marco de pruebas, seleccione XCTest. A continuación, seleccione Elegir archivo.

10. Desplácese hasta el archivo `.zip` que contiene el paquete de XCTest de su aplicación y cárguelo.
11. Una vez que finalice el proceso de carga, seleccione Siguiente.
12. Complete los demás pasos del proceso de creación del proyecto. Seleccionará los dispositivos en los que desea hacer las pruebas y especificará el estado del dispositivo.
13. Después de configurar la ejecución, en la página Revisar e iniciar ejecución, seleccione Confirmar e iniciar ejecución.

    Device Farm ejecuta su prueba y muestra los resultados en la consola.

Trabajar con el marco de pruebas XCTest UI para iOS y AWS Device Farm

Device Farm es compatible con el marco de pruebas XCTest UI para iOS. En concreto, Device Farm admite las pruebas de XCTest UI escritas en Objective-C y en Swift.

Temas

¿Qué es el marco de pruebas XCTest UI?

El marco XCTest UI es el nuevo marco de pruebas introducido con Xcode 7. Este marco extiende las prestaciones de XCTest añadiéndole capacidades para probar interfaces de usuario. Para obtener más información, consulte User Interface Testing en la iOS Developer Library.

Preparación de las pruebas de XCTest UI para iOS

El paquete de pruebas de XCTest UI para iOS debe incluirse en un archivo .ipa con el formato adecuado.

Para crear un archivo.ipa, coloque el paquete my-project-name UITEST-RUNNER.app en un directorio Payload vacío. Después, comprima el directorio Payload en un archivo .zip y, a continuación, cambie la extensión del archivo a .ipa. Xcode produce el paquete *UITest-Runner.app al compilar el proyecto para realizar las pruebas. Se encuentra en el directorio Products del proyecto.

Cargar las pruebas de XCTest UI para iOS

Utilice la consola de Device Farm para cargar las pruebas.

1. Inicie sesión en la consola de Device Farm en https://console.aws.amazon.com/devicefarm.
2. En el panel de navegación de Device Farm, seleccione Pruebas de dispositivos móviles y, a continuación, seleccione Proyectos.
3. En la lista de proyectos, seleccione el proyecto en el que desea cargar las pruebas.

   > **Tip**
   >
   > Puede utilizar la barra de búsqueda para filtrar la lista de proyectos por nombre. Para crear un proyecto, siga las instrucciones de Crear un proyecto en AWS Device Farm.

4. Si se muestra el botón Crear una nueva ejecución, selecciónelo.
5. En la página Elegir aplicación, seleccione Elegir archivo.
6. Busque y elija el archivo de aplicación de iOS. El archivo debe ser un archivo .ipa.

   > **Note**
   >
   > Asegúrese de que el archivo .ipa se ha compilado para un dispositivo iOS y no para un simulador.

7. Seleccione Siguiente.
8. En la página Configurar, en la sección Configurar marco de pruebas, seleccione XCTest UI y, a continuación, seleccione Elegir archivo.
9. Busque y elija el archivo .ipa que contiene la aplicación de ejecución de pruebas de XCTest UI para iOS.
10. Seleccione Siguiente y, después, complete el resto de las instrucciones para seleccionar los dispositivos en los que ejecutar las pruebas e iniciar la ejecución.

Realizar capturas de pantalla en las pruebas de XCTest UI para iOS

Las pruebas de XCTest UI capturan capturas de pantalla de forma automática para cada paso de la pruebas. Estas capturas de pantalla se muestran en el informe de las pruebas de Device Farm. No se requiere código adicional.

Trabajar con pruebas de aplicaciones web en AWS Device Farm

Device Farm ofrece pruebas con Appium para aplicaciones web. Para obtener más información sobre cómo configurar las pruebas de Appium en Device Farm, consulte. the section called “Appium”

Reglas para dispositivos con y sin medidor

La medición hace referencia a la facturación para dispositivos. De forma predeterminada, se realiza una medición de los dispositivos de Device Farm y se le cobrará por minuto después de que consuma los minutos de evaluación gratuitos. También puede optar por adquirir dispositivos sin medidor, lo que le permite realizar un número ilimitado de pruebas por una tarifa plana mensual. Para obtener más información, consulte los precios de AWS Device Farm.

Si decide comenzar una ejecución con un grupo de dispositivos que contenga tanto dispositivos iOS como Android, existen reglas para los dispositivos con y sin medidor. Por ejemplo, si tiene cinco dispositivos Android sin medidor y cinco dispositivos iOS sin medidor, las ejecuciones de pruebas web utilizarán los dispositivos sin medidor.

Este es otro ejemplo: supongamos que tiene cinco dispositivos Android sin medidor y 0 dispositivos iOS sin medidor. Si selecciona solo dispositivos Android para la ejecución web, se usarán los dispositivos sin medidor. Si selecciona tanto dispositivos Android como iOS para la ejecución web, el método de facturación se medirá y no se usarán los dispositivos sin medidor.

Uso de pruebas integradas en AWS Device Farm

Device Farm es compatible con diversos tipos de pruebas integradas para dispositivos Android e iOS.

Tipos de pruebas integradas

Las pruebas integradas permiten probar las aplicaciones sin tener que escribir scripts.

- Integrado: fuzzing (Android e iOS)

Trabajar con la prueba de difusión integrada para Device Farm

Device Farm proporciona un tipo de prueba de difusión integrada.

¿Qué es la prueba de difusión integrada?

La prueba de difusión integrada envía de forma aleatoria eventos de interfaz de usuario a los dispositivos y, a continuación, crea un informe con los resultados.

Uso del tipo de prueba de difusión integrada

Utilice la consola de Device Farm para ejecutar la prueba de difusión integrada.

1. Inicie sesión en la consola de Device Farm en https://console.aws.amazon.com/devicefarm.
2. En el panel de navegación de Device Farm, seleccione Pruebas de dispositivos móviles y, a continuación, seleccione Proyectos.
3. En la lista de proyectos, seleccione el proyecto en el que desea ejecutar la prueba de difusión integrada.

   > **Tip**
   >
   > Puede utilizar la barra de búsqueda para filtrar la lista de proyectos por nombre. Para crear un proyecto, siga las instrucciones de Crear un proyecto en AWS Device Farm.

4. Si se muestra el botón Crear una nueva ejecución, selecciónelo.
5. En la página Elegir aplicación, seleccione Elegir archivo.
6. Busque y elija el archivo de la aplicación en la que desea ejecutar la prueba de difusión integrada.
7. Seleccione Siguiente.
8. En la página Configurar, en la sección Configurar marco de pruebas, seleccione Integrado: fuzzing.
9. Si aparece cualquiera de los siguientes ajustes, puede aceptar los valores predeterminados o especificar los suyos propios:

   - Recuento de eventos: especifique un número entre 1 y 10 000, que representa el número de eventos de interfaz de usuario que va a realizar la prueba de difusión.
   - Acelerador de eventos: especifique un número entre 0 y 1000, que representa el número de milisegundos que debe esperar la prueba de difusión antes de realizar el siguiente evento de interfaz de usuario.

- Semilla aleatorizadora: especifique un número que usará la prueba de difusión para aleatorizar los eventos de interfaz de usuario. Si se especifica el mismo número en las subsiguientes pruebas de difusión, las secuencias de eventos serán idénticas.

10. Seleccione Siguiente y, a continuación, complete el resto de instrucciones para seleccionar dispositivos e inicie la ejecución.

Trabajo con entornos de pruebas personalizados en AWS Device Farm

AWS Device Farm permite configurar un entorno personalizado para las pruebas automatizadas (modo personalizado), que es el enfoque recomendado para todos los usuarios de Device Farm. Para obtener más información sobre los entornos de Device Farm, consulte Test environments (Entornos de prueba).

Entre las ventajas del modo personalizado, en comparación con el modo estándar, se incluyen las siguientes:

- Ejecución end-to-end de pruebas más rápida: el paquete de pruebas no se analiza para detectar todas las pruebas de la suite, lo que evita la sobrecarga de preprocesamiento o posprocesamiento.
- Registro y transmisión de vídeo en directo: los registros de pruebas y el vídeo del lado del cliente se transmiten en directo cuando se utiliza el modo personalizado. Esta característica no está disponible en el modo estándar.
- Captura todos los artefactos: en el host y el dispositivo, el modo personalizado le permite capturar todos los artefactos de prueba. Es posible que esto no sea posible en el modo estándar.
- Entorno local más coherente y replicable: en el modo estándar, se proporcionarán artefactos para cada prueba individual por separado, lo que puede resultar beneficioso en determinadas circunstancias. Sin embargo, el entorno de pruebas local puede diferir de la configuración original, ya que Device Farm gestiona cada prueba ejecutada de forma diferente.

  Por el contrario, el modo personalizado le permite hacer que su entorno de ejecución de pruebas de Device Farm esté en línea de forma coherente con su entorno de pruebas local.

Los entornos personalizados se configuran mediante un archivo de especificaciones de prueba (especificaciones de prueba) con formato YAML. Device Farm proporciona un archivo de especificaciones de prueba predeterminado para cada tipo de prueba compatible que se puede usar tal cual o se puede personalizar; se pueden añadir personalizaciones como filtros de prueba o archivos de configuración a la especificación de prueba. Las especificaciones de prueba editadas se pueden guardar para futuras pruebas.

Para obtener más información, consulte Carga de una especificación de prueba personalizada conAWS CLI y Crear una ejecución de prueba en Device Farm.

Temas

Sintaxis de la especificación de prueba

La estructura del archivo YAML de especificación de prueba es la siguiente:

```
version: 0.1

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

artifacts:
  - location
  - location
```

La especificación de prueba contiene lo siguiente:

`version`

Refleja la versión de la especificación de prueba de Device Farm compatible. El número de versión actual es 0.1.

`phases`

Esta sección contiene grupos de comandos que se ejecutan durante una ejecución de prueba.

Los nombres de las fases de prueba permitidos son:

`install`

Opcional.

Las dependencias predeterminadas de los marcos de pruebas compatibles con Device Farm ya están instaladas. Esta fase contiene los comandos adicionales, si procede, que Device Farm ejecuta durante la instalación.

`pre_test`

Opcional.

Comandos, si procede, que se ejecutan antes de la ejecución de prueba automatizada.

`test`

Opcional.

Comandos que se ejecutan durante de la ejecución de prueba automatizada. Si se produce un error en cualquiera de los comandos de la fase de prueba, la prueba se marca como no completada correctamente.

`post_test`

Opcional.

Comandos, si procede, que se ejecutan después de la ejecución de prueba automatizada.

`artifacts`

Opcional.

Device Farm recopila artefactos, como informes personalizados, archivos de registro e imágenes, de una ubicación que se especifica aquí. No se admiten los caracteres comodín dentro de la ubicación de un artefacto. Por consiguiente, debe especificar una ruta válida para cada ubicación.

Estos artefactos de prueba están disponibles para cada dispositivo de la ejecución de prueba. Para obtener información acerca de la recuperación de artefactos de prueba, consulte Uso de artefactos en un entorno de pruebas personalizado.

> **Important**
>
> Una especificación de prueba tener formato de archivo YAML válido. Si las sangrías o el espaciado de la especificación de prueba no son válidos, la ejecución de prueba puede no completarse correctamente. No se permiten los tabuladores en los archivos YAML. Puede utilizar un validador de YAML para comprobar si la especificación de prueba es un archivo YAML válido. Para obtener más información, consulte el sitio web de YAML.

Ejemplo de especificación de prueba

Este es un ejemplo de una especificación de prueba YAML en Device Farm que configura una ejecución de prueba de Appium Java TestNG:

```
version: 0.1

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

      # This code will wait until the Appium server starts.
```

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

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

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

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

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

```
  - $DEVICEFARM_LOG_DIR
```

Uso del entorno de pruebas de Amazon Linux 2 para pruebas de Android

AWS Device Farm utiliza máquinas host de Amazon Elastic Compute Cloud (EC2) que ejecutan Amazon Linux 2 para ejecutar pruebas de Android. Al programar una ejecución de prueba, Device Farm asigna un host dedicado a cada dispositivo para ejecutar las pruebas de forma independiente. Las máquinas host finalizan después de la ejecución de la prueba, junto con cualquier artefacto generado.

El host de pruebas de Amazon Linux 2 es el entorno de pruebas de Android más reciente y sustituye al anterior sistema basado en Ubuntu. Con el archivo de especificaciones de las pruebas, puede optar por ejecutar las pruebas de Android en el entorno Amazon Linux 2.

El host Amazon Linux 2 ofrece varias ventajas:

- Pruebas más rápidas y fiables: en comparación con el host anterior, el nuevo host de pruebas mejora significativamente la velocidad de las pruebas, lo que reduce especialmente los tiempos de inicio de las pruebas. El host Amazon Linux 2 también demuestra una mayor estabilidad y fiabilidad durante las pruebas.
- Acceso remoto mejorado para las pruebas manuales: las actualizaciones al último host de pruebas y las mejoras permiten reducir la latencia y mejorar el rendimiento del vídeo en las pruebas manuales de Android.
- Selección de versiones de software estándar: Device Farm ahora estandariza el soporte de los principales lenguajes de programación en el host de prueba, así como en las versiones del marco de Appium. Para los lenguajes compatibles (actualmente Java, Python, Node.js y Ruby) y Appium, el nuevo host de pruebas ofrece versiones estables a largo plazo poco después del lanzamiento. La administración centralizada de versiones a través de la herramienta `devicefarm-cli` permite desarrollar archivos con especificaciones de prueba con una experiencia uniforme en todos los marcos.

Temas

- Selección de host de prueba de Android
- Ejemplo de especificación de prueba.
- Migración al host de prueba de Amazon Linux 2

Software compatible

El host de pruebas Amazon Linux 2 viene preinstalado con muchas de las bibliotecas de software necesarias para admitir los marcos de prueba de Device Farm, lo que proporciona un entorno de pruebas listo para el lanzamiento. Para cualquier otro software necesario, puede modificar el archivo de especificaciones de prueba para instalarlo desde su paquete de prueba, descargarlo de Internet o acceder a fuentes privadas dentro de su VPC (consulte VPC ENI para obtener más información). Para obtener más información, consulte el ejemplo del archivo de especificaciones de prueba.

Las siguientes versiones de software están disponibles actualmente en el host:

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

	18	`devicefarm-cli utiliza el nodo 18`
Ruby	2.7	`devicefarm-cli usa ruby 2.7`
	3.2	`devicefarm-cli usa ruby 3.2`
Appium	1	`devicefarm-cli usa appium 1`
	2	`devicefarm-cli utiliza appium 2`

El host de pruebas también incluye herramientas de soporte de uso común para cada versión de software, como los administradores de paquetes `pip` y `npm` (incluidos con Python y Node.js respectivamente) y las dependencias (como el controlador Appium UIAutomator2) para herramientas como Appium. Esto garantiza que dispone de las herramientas necesarias para trabajar con los marcos de prueba compatibles.

La herramienta **`devicefarm-cli`**

El host de pruebas de Amazon Linux 2 utiliza una herramienta de administración de versiones estandarizada llamada `devicefarm-cli` para seleccionar las versiones de software. Esta herramienta es independiente de Device Farm Test Host AWS CLI y solo está disponible en él. Con `devicefarm-cli`, puede cambiar a cualquier versión de software preinstalada en el host de prueba. Esto proporciona una forma sencilla de mantener su archivo de especificaciones de prueba de Device Farm a lo largo del tiempo y le proporciona un mecanismo predecible para actualizar las versiones de software en el futuro.

El siguiente fragmento muestra la página `help` de `devicefarm-cli`:

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

     Commands:
         help                        Prints this usage message.
         list                        Lists all versions of software configurable
```

```
                                    via this CLI.
        use <software> <version>    Configures the software for usage within the
                                    current shell's environment.
```

Repasemos un par de ejemplos que utilizan `devicefarm-cli`. Para usar la herramienta para cambiar la versión de Python de *`3.10`* a *`3.9`* en su archivo de especificaciones de prueba, ejecute los siguientes comandos:

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

Para cambiar de la versión *`1`* a *`2`* de Appium:

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

> **Tip**
>
> Tenga en cuenta que cuando selecciona una versión de software, `devicefarm-cli` también cambia las herramientas compatibles con esos lenguajes, como `pip` para Python y `npm` para Nodejs.

Selección de host de prueba de Android

Para las pruebas de Android, Device Farm requiere el siguiente campo en el archivo de especificaciones de la prueba para elegir el host de prueba de Amazon Linux 2:

```
android_test_host: amazon_linux_2 | legacy
```

Use `amazon_linux_2` para ejecutar las pruebas en el host de pruebas de Amazon Linux 2:

```
android_test_host: amazon_linux_2
```

Obtenga más información sobre las ventajas de Amazon Linux 2 aquí.

Device Farm recomienda usar el host Amazon Linux 2 para las pruebas de Android en lugar del entorno de host heredado. Si prefiere usar el entorno heredado, use`legacy` para ejecutar las pruebas en el host de pruebas heredado:

```
android_test_host: legacy
```

De forma predeterminada, los archivos de especificaciones de prueba sin una selección de host de prueba se ejecutarán en el host de prueba heredado.

Sintaxis obsoleta

A continuación se muestra la sintaxis obsoleta para elegir Amazon Linux 2 en el archivo de especificaciones de prueba:

```
preview_features:
  android_amazon_linux_2_host: true
```

Si utiliza este indicador, las pruebas seguirán ejecutándose en Amazon Linux 2. Sin embargo, recomendamos encarecidamente eliminar la sección de banderas `preview_features` y sustituirla por el nuevo campo `android_test_host` para evitar gastos de mantenimiento en el futuro.

> **Warning**
>
> Si utiliza los indicadores `android_test_host` y `android_amazon_linux_2_host` en el archivo de especificaciones de la prueba, se generará un error. Solo se debe usar uno; recomendamos `android_test_host`.

Ejemplo de especificación de prueba.

El siguiente fragmento es un ejemplo de un archivo de especificaciones de prueba de Device Farm que configura una ejecución de prueba de NodeJS de Appium con el host de pruebas Amazon Linux 2 para Android:

```
version: 0.1

# This flag enables your test to run using Device Farm's Amazon Linux 2 test host. For
 more information,
```

```
# please see https://docs.aws.amazon.com/devicefarm/latest/developerguide/amazon-
linux-2.html
android_test_host: amazon_linux_2

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

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

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

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

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

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

```
      # For Appium version 2, Device Farm automatically updates the preinstalled
 UIAutomator2 driver
      # over time to incorporate the latest minor and patch versions for its major
 version 2. If you
      # want to install a specific version of the driver, you can use the Appium
 extension CLI to
      # uninstall the existing UIAutomator2 driver and install your desired version:
      # - appium driver uninstall uiautomator2
      # - appium driver install uiautomator2@2.34.0

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

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

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

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

      # We recommend starting the Appium server process in the background using the
 command below.
      # The Appium server log will be written to the $DEVICEFARM_LOG_DIR directory.
```

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

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

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

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

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

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

Migración al host de prueba de Amazon Linux 2

Para migrar las pruebas existentes del host anterior al nuevo host de Amazon Linux 2, desarrolle nuevos archivos de especificaciones de pruebas basados en los ya existentes. El enfoque recomendado es comenzar con los nuevos archivos de especificaciones de prueba predeterminados para sus tipos de prueba. A continuación, migre los comandos pertinentes del archivo de especificaciones de prueba anterior al nuevo y guarde el archivo anterior como copia de seguridad. Esto le permite aprovechar la especificación predeterminada optimizada para el nuevo host y, al mismo tiempo, reutilizar el código existente. Garantiza que obtendrá todos los beneficios del nuevo host configurado de manera óptima para sus pruebas y, al mismo tiempo, conservará las especificaciones de prueba antiguas como referencia a la hora de adaptar los comandos al nuevo entorno.

Puede seguir los siguientes pasos para crear un nuevo archivo de especificaciones de prueba de Amazon Linux 2 y, al mismo tiempo, reutilizar los comandos del archivo de especificaciones de prueba anterior:

1. Inicie sesión en la consola de Device Farm en https://console.aws.amazon.com/devicefarm.

2. Navegue hasta el proyecto Device Farm que contiene sus pruebas de automatización.
3. Seleccione Crear una nueva ejecución de prueba en el proyecto.
4. Seleccione una aplicación y un paquete de pruebas utilizados anteriormente para su marco de pruebas.
5. Seleccione Ejecutar la prueba en un entorno personalizado.
6. Seleccione el archivo de especificaciones de prueba que está utilizando actualmente para las pruebas en el host de pruebas anterior en el menú desplegable de especificaciones de prueba.
7. Copie el contenido de este archivo y péguelo localmente en un editor de texto para consultarlo más adelante.
8. En el menú desplegable de especificaciones de prueba, cambie la selección de especificaciones de prueba por el archivo de especificaciones de prueba predeterminado más reciente.
9. Seleccione Editar y entrará en la interfaz de edición de especificaciones de prueba. Observará que, en las primeras líneas del archivo de especificaciones de la prueba, ya ha dado de alta el nuevo host de la prueba:

```
android_test_host: amazon_linux_2
```

10. Revise la sintaxis para seleccionar los hosts de prueba aquí y las principales diferencias entre los hosts de prueba aquí.
11. Añada y edite de forma selectiva los comandos del archivo de especificaciones de prueba guardado localmente desde el paso 6 al nuevo archivo de especificaciones de prueba predeterminado. A continuación, seleccione Guardar como para guardar el nuevo archivo de especificaciones. Ahora puede programar las pruebas en el host de pruebas de Amazon Linux 2.

Diferencias entre los hosts de prueba nuevos y antiguos

Al editar el archivo de especificaciones de prueba para utilizar el host de pruebas de Amazon Linux 2 y al realizar la transición de las pruebas desde el host de pruebas anterior, tenga en cuenta estas diferencias clave entre los entornos:

- Selección de versiones de software: en muchos casos, las versiones de software predeterminadas han cambiado, por lo que si antes no seleccionó explícitamente su versión de software en el host de pruebas Legacy, puede que desee especificarla ahora en el host de pruebas de Amazon Linux 2 utilizando `devicefarm-cli`. En la gran mayoría de los casos de uso, recomendamos que los clientes seleccionen de forma explícita las versiones del software que utilizan. Si selecciona una versión de software con `devicefarm-cli`, tendrá una experiencia predecible y coherente y

recibirá una gran cantidad de advertencias si Device Farm planea eliminar esa versión del host de prueba.

Además, herramientas de selección de software como `nvm`, `pyenv`, `avm`, y `rvm` se han eliminado en favor del nuevo sistema de selección de software `devicefarm-cli`.

- Versiones de software disponibles: se han eliminado muchas versiones del software previamente preinstalado y se han añadido muchas versiones nuevas. Por lo tanto, asegúrese de que cuando utilice `devicefarm-cli` para seleccionar las versiones de software, seleccione las que estén en la lista de versiones compatibles.
- Las rutas de archivo con codificación rígida en el archivo de especificaciones de prueba de su host heredado probablemente no funcionen según lo previsto en el host de pruebas de Amazon Linux 2; por lo general, no se recomiendan para el uso de archivos de especificaciones de prueba. Le recomendamos que utilice rutas relativas y variables de entorno para todo el código del archivo de especificaciones de prueba. Además, tenga en cuenta que la mayoría de los binarios que necesita para la prueba se encuentran en la ruta del host, de modo que se puedan ejecutar inmediatamente desde el archivo de especificaciones utilizando solo su nombre (por ejemplo, Appium).
- Por el momento, el nuevo host de prueba no admite la recopilación de datos de rendimiento.
- Versión del sistema operativo: el host de prueba anterior estaba basado en el sistema operativo Ubuntu, mientras que el nuevo se basa en Amazon Linux 2. Como resultado, los usuarios pueden observar algunas diferencias en las bibliotecas del sistema disponibles y en las versiones de las bibliotecas del sistema.
- Para los usuarios de Appium Java, el nuevo host de prueba no contiene ningún archivo JAR preinstalado en su ruta de clases, mientras que el host anterior contenía uno para el marco TestNG (a través de una variable de entorno). `$DEVICEFARM_TESTNG_JAR` Recomendamos a los clientes que incluyan los archivos JAR necesarios para sus marcos de pruebas dentro de su paquete de pruebas y que eliminen las instancias de la variable `$DEVICEFARM_TESTNG_JAR` de sus archivos de especificaciones de prueba. Para obtener más información, consulte Uso de Appium y AWS Device Farm.
- Para los usuarios de Appium, se ha eliminado la variable de entorno `$DEVICEFARM_CHROMEDRIVER_EXECUTABLE` en favor de un nuevo enfoque que permite a los clientes acceder a Chromedriver para Android. Consulte nuestro Archivo de especificaciones de prueba predeterminado para ver un ejemplo, que usa una nueva variable del entorno `$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR`.

> **Note**
>
> Recomendamos encarecidamente mantener el comando del servidor Appium existente del archivo de especificaciones de prueba predeterminado tal como está.

Le recomendamos que contacte con el equipo de servicio a través de un servicio de asistencia si tiene algún comentario o pregunta sobre las diferencias entre los hosts de prueba desde el punto de vista del software.

Variables de entorno

Las variables de entorno representan los valores que se utilizan en las pruebas automatizadas. Puede utilizar estas variables de entorno en los archivos YAML y en el código de prueba. En un entorno de pruebas personalizado, Device Farm rellena forma dinámica las variables de entorno en tiempo de ejecución.

Temas

Variables de entorno comunes

Pruebas de Android

En esta sección se describen las variables de entorno personalizadas comunes para las pruebas de la plataforma Android compatibles con Device Farm.

`$DEVICEFARM_DEVICE_NAME`

Nombre del dispositivo en el que se ejecutan las pruebas. Representa el UDID (identificador único de dispositivo) del dispositivo.

`$DEVICEFARM_DEVICE_PLATFORM_NAME`

Nombre de la plataforma del dispositivo. Puede ser Android o iOS.

$DEVICEFARM_DEVICE_OS_VERSION

La versión del sistema operativo del dispositivo.

$DEVICEFARM_APP_PATH

La ruta a la aplicación móvil en la máquina host donde se ejecutan las pruebas. La ruta de la aplicación solo está disponible para aplicaciones móviles.

$DEVICEFARM_DEVICE_UDID

Identificador único del dispositivo móvil en el que se ejecuta la prueba automatizada.

$DEVICEFARM_LOG_DIR

Ruta a los archivos de registro generados durante la ejecución de prueba. De forma predeterminada, todos los archivos de este directorio se archivan en un archivo ZIP y están disponibles como artefactos tras la ejecución de la prueba.

$DEVICEFARM_SCREENSHOT_PATH

Ruta a las capturas de pantalla, si procede, capturadas durante la ejecución de prueba.

$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR

La ubicación de un directorio que contiene los ejecutables de Chromedriver necesarios para su uso en las pruebas web e híbridas de Appium.

$ANDROID_HOME

La ruta al directorio de instalación del SDK de Android.

> **Note**
>
> La variable del entorno `ANDROID_HOME` solo está disponible en el host de pruebas de Amazon Linux 2 para Android.

Pruebas de iOS

En esta sección se describen las variables de entorno personalizadas comunes para las pruebas de la plataforma iOS compatibles con Device Farm.

$DEVICEFARM_DEVICE_NAME

Nombre del dispositivo en el que se ejecutan las pruebas. Representa el UDID (identificador único de dispositivo) del dispositivo.

$DEVICEFARM_DEVICE_PLATFORM_NAME

Nombre de la plataforma del dispositivo. Puede ser Android o iOS.

$DEVICEFARM_APP_PATH

La ruta a la aplicación móvil en la máquina host donde se ejecutan las pruebas. La ruta de la aplicación solo está disponible para aplicaciones móviles.

$DEVICEFARM_DEVICE_UDID

Identificador único del dispositivo móvil en el que se ejecuta la prueba automatizada.

$DEVICEFARM_LOG_DIR

Ruta a los archivos de registro generados durante la ejecución de prueba.

$DEVICEFARM_SCREENSHOT_PATH

Ruta a las capturas de pantalla, si procede, capturadas durante la ejecución de prueba.

Variables de entorno de Appium Java JUnit

En esta sección se describen las variables de entorno que se utilizan en las pruebas de Appium Java JUnit en un entorno de pruebas personalizado.

$DEVICEFARM_TESTNG_JAR

Ruta al archivo .jar de TestNG.

$DEVICEFARM_TEST_PACKAGE_PATH

Ruta al contenido descomprimido del archivo del paquete de prueba.

Variables de entorno de Appium Java TestNG

En esta sección se describen las variables de entorno que se utilizan en las pruebas de Appium Java TestNG en un entorno de pruebas personalizado.

$DEVICEFARM_TESTNG_JAR

Ruta al archivo .jar de TestNG.

$DEVICEFARM_TEST_PACKAGE_PATH

Ruta al contenido descomprimido del archivo del paquete de prueba.

Variables de entorno de XCUITest

$DEVICEFARM_XCUITESTRUN_FILE

Ruta al archivo `.xctestun` de Device Farm. Se genera a partir de los paquetes de prueba y de la aplicación.

$DEVICEFARM_DERIVED_DATA_PATH

Ruta esperada de salida de xcodebuild de Device Farm.

Migración de pruebas de un entorno de pruebas estándar a un entorno de pruebas personalizado

La siguiente guía explica cómo cambiar de un modo de ejecución de pruebas estándar a un modo de ejecución personalizado. La migración implica principalmente dos formas diferentes de ejecución:

1. Modo estándar: este modo de ejecución de pruebas está diseñado principalmente para proporcionar a los clientes informes detallados y un entorno totalmente gestionado.
2. Modo personalizado: este modo de ejecución de pruebas está diseñado para diferentes casos de uso que requieren ejecuciones de pruebas más rápidas, la capacidad de migrar mediante lift-and-shift y lograr la paridad con su entorno local y la transmisión de vídeo en directo.

Consideraciones a la hora de migrar

En esta sección se enumeran algunos de los casos de uso más destacados que se deben tener en cuenta al migrar al modo personalizado:

1. Velocidad: en el modo de ejecución estándar, Device Farm analiza los metadatos de las pruebas que ha empaquetado y cargado siguiendo las instrucciones de empaquetado de su marco

particular. El análisis detecta el número de pruebas del paquete. A partir de entonces, Device Farm ejecuta cada prueba por separado y presenta los registros, vídeos y otros artefactos de resultados de forma individual para cada prueba. Sin embargo, esto aumenta constantemente el tiempo total de ejecución de las end-to-end pruebas, ya que, por parte del servicio, se producen alteraciones en el procesamiento previo y posterior de las pruebas y en los resultados.

Por el contrario, el modo de ejecución personalizado no analiza el paquete de pruebas, lo que significa que las pruebas o los resultados se producen sin preprocesamiento y con un mínimo de posprocesamiento. Esto se traduce en tiempos totales end-to-end de ejecución cercanos a los de su configuración local. Las pruebas se ejecutan en el mismo formato que si se ejecutaran en los equipos locales. Los resultados de las pruebas son los mismos que los que se obtienen localmente y están disponibles para su descarga al final de la ejecución del trabajo.

2. Personalización o flexibilidad: el modo de ejecución estándar analiza el paquete de pruebas para detectar el número de pruebas y, a continuación, ejecuta cada prueba por separado. Tenga en cuenta que no hay garantía de que las pruebas se ejecuten en el orden que especificó. Como resultado, es posible que las pruebas que requieren una secuencia de ejecución determinada no funcionen según lo esperado. Además, no hay forma de personalizar el entorno de la máquina host ni de pasar los archivos de configuración que puedan ser necesarios para ejecutar las pruebas de una forma determinada.

   Por el contrario, el modo personalizado le permite configurar el entorno de la máquina host, incluida la posibilidad de instalar software adicional, pasar filtros a las pruebas, transferir archivos de configuración y controlar la configuración de ejecución de las pruebas. Lo consigue mediante un archivo yaml (también denominado archivo testspec) que puede modificar añadiéndole comandos de intérprete de comandos. Este archivo yaml se convierte en un script de intérprete de comandos que se ejecuta en el equipo host de prueba. Puede guardar varios archivos yaml y elegir uno de forma dinámica según sus necesidades al programar una ejecución.

3. Vídeo en directo y registro: los modos de ejecución estándar y personalizado le proporcionan vídeos y registros para sus pruebas. Sin embargo, en el modo estándar, solo obtendrá el vídeo y los registros predefinidos de las pruebas una vez finalizadas las pruebas.

   Por el contrario, el modo personalizado te ofrece una transmisión en directo del vídeo y de los registros de sus pruebas desde el lado del cliente. Además, podrá descargar el vídeo y otros artefactos al final de las pruebas.

4. Obsolación: los siguientes tipos de pruebas quedarán obsoletos a finales de diciembre de 2023 en el modo de ejecución estándar:

   - Appium (todos los idiomas)

- Calabash
- XCTest
- UI Automation
- UI Automator
- Pruebas web
- Built-in: Explorer

Una vez que estén en desuso, no podrá utilizar estos marcos en modo estándar. En su lugar, puede utilizar el modo personalizado para los tipos de prueba enumerados anteriormente.

> **Tip**
>
> Si su caso de uso implica al menos uno de los factores anteriores, le recomendamos encarecidamente que cambie al modo de ejecución personalizado.

Pasos para realizar la migración

Para migrar del modo estándar al modo personalizado, haga lo siguiente:

1. Inicie sesión en la consola Device Farm AWS Management Console y ábrala en https://console.aws.amazon.com/devicefarm/.
2. Seleccione su proyecto y, a continuación, inicie una nueva ejecución de automatización.
3. Cargue su aplicación (o `web app` selecciónela), seleccione el tipo de marco de prueba, cargue su paquete de prueba y, a continuación, en el parámetro `Choose your execution environment`, seleccione la opción para `Run your test in a custom environment`.
4. De forma predeterminada, aparecerá el ejemplo del archivo de especificaciones de prueba de Device Farm para que lo vea y lo edite. Este archivo de ejemplo se puede utilizar como punto de partida para probar las pruebas en el modo de entorno personalizado. A continuación, una vez que haya comprobado que las pruebas funcionan correctamente desde la consola, podrá modificar cualquiera de sus integraciones de API, CLI y canalización con Device Farm para utilizar este archivo de especificaciones de prueba como parámetro al programar las ejecuciones de las pruebas. Para obtener información sobre cómo añadir un archivo de especificaciones de prueba como parámetro para tus ejecuciones, consulte la sección de parámetros de `testSpecArn` de la API `ScheduleRun` en nuestra guía de API.

Marco de Appium

En un entorno de pruebas personalizado, Device Farm no inserta ni anula ninguna capacidad de Appium en las pruebas del marco de Appium. Es preciso especificar las capacidades de Appium de la prueba en el archivo YAML de la especificación de prueba o en el código de la prueba.

Instrumentación para Android

No es preciso realizar cambios para mover las pruebas de instrumentación para Android a un entorno de pruebas personalizado.

iOS XCUITest

No es preciso realizar cambios para mover las pruebas de iOS XCUITest a un entorno de pruebas personalizado.

Ampliación de los entornos de prueba personalizados en Device Farm

El modo personalizado de Device Farm le permite ejecutar algo más que su conjunto de pruebas. En esta sección, aprenderá a ampliar su conjunto de pruebas y a optimizar sus pruebas.

Configurar un PIN

Algunas aplicaciones requieren que establezca un PIN en el dispositivo. Device Farm no admite la configuración de un PIN en los dispositivos de forma nativa. Sin embargo, esto es posible con las siguientes advertencias:

- El dispositivo debe ejecutar Android 8 o superior.
- El PIN debe eliminarse una vez finalizada la prueba.

Para configurar el PIN en las pruebas, utilice las fases `pre_test` y `post_test` para configurar y eliminar el PIN, tal y como se muestra a continuación:

```
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"
```

Cuando comience el conjunto de pruebas, se establecerá el PIN 1234. Al salir de la sala de pruebas, se elimina el PIN.

> **Warning**
>
> Si no elimina el PIN del dispositivo una vez finalizada la prueba, el dispositivo y su cuenta se pondrán en cuarentena.

Agilizar las pruebas basadas en Appium con las capacidades deseadas

Cuando se utiliza Appium, se puede encontrar que el conjunto de pruebas de modo estándar es muy lento. Esto se debe a que Device Farm aplica la configuración predeterminada y no hace suposiciones sobre cómo desea utilizar el entorno de Appium. Si bien estos valores predeterminados se basan en las prácticas recomendadas del sector, es posible que no se apliquen a su situación. Para ajustar los parámetros del servidor Appium, puede ajustar las capacidades predeterminadas de Appium en sus especificaciones de prueba. Por ejemplo, lo siguiente establece la capacidad de `usePrebuildWDA` en `true` de un conjunto de pruebas de iOS para acelerar el tiempo de inicio 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 &
```

Las capacidades de Appium deben ser una estructura JSON entrecomillada con intérprete de comandos de escape.

Las siguientes capacidades de Appium son fuentes comunes de mejoras en el rendimiento:

`noReset` y `fullReset`

Estas dos capacidades, que se excluyen mutuamente, describen el comportamiento de Appium una vez finalizada cada sesión. Cuando `noReset` está configurado en la opción `true`, el servidor de Appium no elimina los datos de la aplicación al finalizar una sesión de Appium, por lo que no realiza ningún tipo de limpieza. `fullReset`desinstala y borra todos los datos de la aplicación del dispositivo una vez cerrada la sesión. Para obtener más información, consulte Restablecer estrategias en la documentación de Appium.

`ignoreUnimportantViews` (Solo para Android)

Indica a Appium que comprima la jerarquía de la interfaz de usuario de Android solo para incluir las vistas relevantes para la prueba, lo que acelera las búsquedas de ciertos elementos. Sin embargo, esto puede afectar a algunos conjuntos de pruebas basados en XPath porque se ha modificado la jerarquía del diseño de la interfaz de usuario.

`skipUnlock` (Solo para Android)

Informa a Appium de que actualmente no hay ningún código PIN configurado, lo que acelera las pruebas después de que se apague la pantalla o se bloquee.

`webDriverAgentUrl` (solo iOS)

Indica a Appium que asuma que una dependencia esencial de iOS, `webDriverAgent`, ya está en ejecución y disponible para aceptar solicitudes HTTP en la URL especificada. Si `webDriverAgent` aún no está en funcionamiento, Appium puede tardar algún tiempo al principio de un conjunto de pruebas en iniciar el `webDriverAgent`. Si inicia `webDriverAgent` usted mismo y configura `webDriverAgentUrl` como `http://localhost:8100` al iniciar Appium, podrá arrancar su conjunto de pruebas más rápido. Tenga en cuenta que esta capacidad nunca debe usarse junto con la capacidad `useNewWDA`.

Puede usar el siguiente código para empezar `webDriverAgent` desde el archivo de especificaciones de prueba en el puerto local del dispositivo `8100` y, a continuación, reenviarlo al puerto local del host de la prueba `8100` (esto te permite establecer el valor de `webDriverAgentUrl` como `http://localhost:8100`). Este código debe ejecutarse durante

la fase de instalación, una vez que se haya definido cualquier código para configurar las variables de Appium y de entorno `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 &
```

Luego, puede añadir el siguiente código a su archivo de especificaciones de prueba para asegurarse de que `webDriverAgent` se haya iniciado correctamente. Este código debe ejecutarse al final de la fase de prueba previa después de garantizar que Appium se haya iniciado correctamente:

```
      # 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 obtener más información sobre las capacidades que admite Appium, consulte las capacidades deseadas de Appium en la documentación de Appium.

Uso de Webhooks y otras API después de ejecutar las pruebas

Puede hacer que Device Farm llame a un webhook después de que cada conjunto de pruebas termine de usar curl. El proceso para hacerlo varía según el destino y el formato. Para su webhook específico, consulte la documentación de ese webhook. En el siguiente ejemplo, se publica un mensaje en un webhook de Slack cada vez que un conjunto de pruebas finaliza:

```
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 obtener más información sobre el uso de webhooks con Slack, consulte Cómo enviar su primer mensaje de Slack mediante Webhook en la referencia de la API de Slack.

No está limitado a utilizar curl para llamar a los webhooks. Los paquetes de prueba pueden incluir scripts y herramientas adicionales, siempre que sean compatibles con el entorno de ejecución de Device Farm. Por ejemplo, su paquete de prueba puede incluir scripts auxiliares que realicen solicitudes a otras API. Asegúrese de que todos los paquetes necesarios estén instalados junto con los requisitos de su conjunto de pruebas. Para añadir un script que se ejecute una vez finalizado el conjunto de pruebas, inclúyalo en el paquete de prueba y añada lo siguiente a las especificaciones de la prueba:

```
phases:
  post_test:
    - python post_test.py
```

> **Note**
>
> El mantenimiento de las claves de API u otros tokens de autenticación utilizados en su paquete de prueba es su responsabilidad. Le recomendamos que mantenga cualquier

tipo de credencial de seguridad fuera del control de código fuente, que utilice credenciales con el menor número de privilegios posible y que utilice tokens revocables y de corta duración siempre que sea posible. Para verificar los requisitos de seguridad, consulte la documentación de las API de terceros que utilice.

Si planea usar los servicios de AWS como parte de su conjunto de ejecución de pruebas, debe usar las credenciales temporales de IAM, generadas fuera de su conjunto de pruebas e incluidas en su paquete de pruebas. Estas credenciales deben tener el menor número de permisos concedidos y una vida útil lo más corta posible. Para obtener más información acerca de la creación de credenciales temporales, consulte Solicitud de credenciales de seguridad temporales en la Guía del usuario de IAM.

Añadir archivos adicionales a su paquete de prueba

Es posible que desee utilizar archivos adicionales como parte de sus pruebas, ya sea como archivos de configuración adicionales o como datos de prueba adicionales. Puede añadir estos archivos adicionales a su paquete de prueba antes de cargarlo en AWS Device Farm y, a continuación, acceder a ellos desde el modo de entorno personalizado. Básicamente, todos los formatos de carga de paquetes de prueba (ZIP, IPA, APK, JAR, etc.) son formatos de archivo de paquetes que admiten las operaciones ZIP estándar.

Puede añadir archivos a su archivo de prueba antes de cargarlos en AWS Device Farm mediante el siguiente comando:

```
$ zip zip-with-dependencies.zip extra_file
```

Para un directorio de archivos adicionales:

```
$ zip -r zip-with-dependencies.zip extra_files/
```

Estos comandos funcionan según lo previsto en todos los formatos de carga de paquetes de prueba, excepto en los archivos IPA. Para los archivos IPA, especialmente cuando se utilizan con XcuiTests, le recomendamos que coloque los archivos adicionales en una ubicación ligeramente diferente debido a la forma en que AWS Device Farm diseña los paquetes de prueba de iOS. Al compilar la prueba de iOS, el directorio de la aplicación de prueba se ubicará dentro de otro directorio llamado *`Payload`*.

Por ejemplo, este es el aspecto que puede tener uno de esos directorios de prueba de iOS:

```
$ tree
.
### Payload
    ### ADFiOSReferenceAppUITests-Runner.app
        ### ADFiOSReferenceAppUITests-Runner
        ### Frameworks
        #   ### XCTAutomationSupport.framework
        #   #   ### Info.plist
        #   #   ### XCTAutomationSupport
        #   #   ### _CodeSignature
        #   #   #   ### CodeResources
        #   #   ### version.plist
        #   ### XCTest.framework
        #       ### Info.plist
        #       ### XCTest
        #       ### _CodeSignature
        #       #   ### CodeResources
        #       ### en.lproj
        #       #   ### InfoPlist.strings
        #       ### version.plist
        ### Info.plist
        ### PkgInfo
        ### PlugIns
        #   ### ADFiOSReferenceAppUITests.xctest
        #   #   ### ADFiOSReferenceAppUITests
        #   #   ### Info.plist
        #   #   ### _CodeSignature
        #   #       ### CodeResources
        #   ### ADFiOSReferenceAppUITests.xctest.dSYM
        #       ### Contents
        #           ### Info.plist
        #           ### Resources
        #               ### DWARF
        #                   ### ADFiOSReferenceAppUITests
        ### _CodeSignature
        #   ### CodeResources
        ### embedded.mobileprovision
```

Para estos paquetes de XCUITest, añada cualquier archivo adicional al directorio que termine en *`.app`* dentro del directorio *`Payload`*. Por ejemplo, los siguientes comandos muestran cómo se puede añadir un archivo a este paquete de prueba:

```
$ mv extra_file Payload/*.app/
$ zip -r my_xcui_tests.ipa Payload/
```

Al añadir un archivo a su paquete de prueba, puede esperar un comportamiento de interacción ligeramente diferente en AWS Device Farm en función del formato de carga. Si la carga utilizó la extensión de archivo ZIP, AWS Device Farm lo descomprimirá automáticamente antes de la prueba y dejará los archivos descomprimidos en la ubicación con la variable de entorno *$DEVICEFARM_TEST_PACKAGE_PATH*. (Esto significa que si añade un archivo llamado *extra_file* a la raíz del archivo, como en el primer ejemplo, se ubicará en *$DEVICEFARM_TEST_PACKAGE_PATH/extra_file* durante la prueba).

Para usar un ejemplo más práctico, si es un usuario de Appium TestNG que quiere incluir un archivo *testng.xml* con su prueba, puede incluirlo en su archivo usando el siguiente comando:

```
$ zip zip-with-dependencies.zip testng.xml
```

Luego, puede cambiar su comando de prueba en el modo de entorno personalizado por el siguiente:

```
java -D appium.screenshots.dir=$DEVICEFARM_SCREENSHOT_PATH org.testng.TestNG -testjar
 *-tests.jar -d $DEVICEFARM_LOG_DIR/test-output $DEVICEFARM_TEST_PACKAGE_PATH/
testng.xml
```

Si la extensión de carga del paquete de prueba no es ZIP (por ejemplo, un archivo APK, IPA o JAR), el archivo del paquete cargado propiamente dicho se encuentra en *$DEVICEFARM_TEST_PACKAGE_PATH*. Como siguen siendo archivos en formato de archivo, puede descomprimirlos para acceder a los archivos adicionales desde dentro. Por ejemplo, el siguiente comando descomprimirá el contenido del paquete de prueba (para archivos APK, IPA o JAR) en el directorio */tmp*:

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

En el caso de un archivo APK o JAR, encontrará los archivos adicionales descomprimidos en el directorio */tmp* (por ejemplo, */tmp/extra_file*). En el caso de un archivo IPA, como se ha explicado anteriormente, los archivos adicionales estarían en una ubicación ligeramente diferente dentro de la carpeta que termina en *.app*, que se encuentra dentro del directorio *Payload*. Por ejemplo, según el ejemplo anterior de IPA, el archivo se encontraría en la ubicación */tmp/payload/ADFIOSReferenceAppuitests-Runner.app/extra_file* (referenciable como */TMP/payload/ *.app/extra_file*).

Trabajar con acceso remoto en AWS Device Farm

El acceso remoto le permite deslizar el dedo por la pantalla, realizar gestos e interactuar con un dispositivo a través del navegador web en tiempo real, con el fin de probar la funcionalidad y reproducir los problemas de los clientes. Usted interactúa con un dispositivo concreto mediante la creación de una sesión de acceso remoto con ese dispositivo.

Una sesión en Device Farm es una interacción en tiempo real con un dispositivo físico real con un navegador web como host. Una sesión muestra el único dispositivo seleccionado al comenzar la sesión. Un usuario puede comenzar más de una sesión a la vez con el número total de dispositivos simultáneos limitado por el número de ranuras de dispositivos que tiene. Puede adquirir ranuras de dispositivos en función de la familia de dispositivos (dispositivos Android o iOS). Para obtener más información, consulte los precios de Device Farm.

En la actualidad, Device Farm ofrece un subconjunto de dispositivos para pruebas de acceso remoto. Continuamente estamos añadiendo nuevos dispositivos al grupo de dispositivos.

Device Farm captura vídeo de cada sesión de acceso remoto y genera registros de la actividad que tiene lugar durante la sesión. Estos resultados incluyen toda la información que usted proporciona durante una sesión.

> **Note**
>
> Por motivos de seguridad, le recomendamos que evite proporcionar o escribir información confidencial como, por ejemplo, números de cuenta, información de inicio de sesión personal y otros detalles durante una sesión de acceso remoto.

Temas

Crear una sesión de acceso remoto en AWS Device Farm

Para obtener información acerca de las sesiones de acceso remoto, consulte Sesiones.

Requisitos previos

- Crear un proyecto en Device Farm. Siga las instrucciones de Crear un proyecto en AWS Device Farm y, a continuación, vuelva a esta página.

Crear una sesión con la consola de Device Farm

1. Inicie sesión en la consola de Device Farm en https://console.aws.amazon.com/devicefarm.
2. En el panel de navegación de Device Farm, seleccione Pruebas de dispositivos móviles y, a continuación, seleccione Proyectos.
3. Si ya tiene un proyecto, selecciónelo de la lista. De lo contrario, para crear un proyecto, siga las instrucciones de Crear un proyecto en AWS Device Farm.
4. En la pestaña Acceso remoto, seleccione Iniciar una nueva sesión.
5. Elija un dispositivo para la sesión. Puede elegir en la lista de dispositivos disponibles o buscar un dispositivo mediante la barra de búsqueda en la parte superior de la lista. Puede buscar por:
   - Nombre
   - Plataforma
   - Factor de forma
   - Tipo de flota
6. En Nombre de sesión, escriba un nombre para la sesión.
7. Seleccione Confirmar e iniciar sesión.

Pasos siguientes

Device Farm inicia la sesión en cuanto el dispositivo solicitado esté disponible, normalmente después de unos minutos. Se muestra el cuadro de diálogo Dispositivo solicitado hasta que comienza la sesión. Para cancelar la solicitud de sesión, seleccione Cancelar solicitud.

Una vez que se ha iniciado una sesión, si debe cerrar el navegador o una pestaña del navegador sin parar la sesión o si se pierde la conexión entre el navegador e Internet, la sesión permanecerá activa durante cinco minutos a partir de ese momento. Después de eso, Device Farm finaliza la sesión. El tiempo de inactividad se le cargará en su cuenta.

Una vez que se ha iniciado la sesión, puede interactuar con el dispositivo en el navegador web.

Usar una sesión de acceso remoto en AWS Device Farm

Para obtener información sobre la realización de pruebas interactivas de aplicaciones Android e iOS a través de sesiones de acceso remoto, consulte Sesiones,

- Requisitos previos
- Uso de una sesión en la consola de Device Farm
- Pasos siguientes
- Sugerencias y trucos

Requisitos previos

- Cree una sesión. Siga las instrucciones de Crear una sesión y, a continuación, vuelva a esta página.

Uso de una sesión en la consola de Device Farm

En cuanto el dispositivo que ha solicitado para una sesión de acceso remoto esté disponible, la consola muestra la pantalla del dispositivo. La sesión tiene una longitud máxima de 150 minutos. El tiempo restante de la sesión aparece en el campo Tiempo restante junto al nombre del dispositivo.

Instalación de una aplicación

Para instalar una aplicación en el dispositivo de la sesión, en Instalar aplicaciones, seleccione Cargar y, a continuación, seleccione el archivo .apk (Android) o el archivo .ipa (iOS) que desee instalar. Las aplicaciones que se ejecutan en una sesión de acceso remoto no requieren aprovisionamiento ni instrumentación de pruebas.

> **Note**
>
> AWS Device Farm no muestra una confirmación después de instalar la aplicación. Pruebe a interactuar con el icono de la aplicación para ver si está lista para su uso.
> Cuando carga una aplicación, a veces existe un retraso antes de que la aplicación esté disponible. Observe la bandeja del sistema para determinar si la aplicación está disponible.

Control del dispositivo

Puede interactuar con el dispositivo que se muestra en la consola del mismo modo que haría con el dispositivo físico real. Para ello, utilice el ratón o un dispositivo equivalente para tocar y el teclado en pantalla del dispositivo. Para dispositivos Android, existen botones en Controles de visualización que funcionan como los botones Inicio y Atrás de un dispositivo Android. Para dispositivos iOS, existe un botón Inicio que funciona exactamente igual que el botón de inicio de un dispositivo iOS. También puede cambiar entre las aplicaciones que se ejecutan en el dispositivo seleccionando Aplicaciones recientes.

Cambio entre los modos vertical y horizontal

También puede cambiar entre los modos vertical y horizontal para los dispositivos que utilice.

Pasos siguientes

Device Farm continúa la sesión hasta que se para manualmente o se alcanza el límite de tiempo de 150 minutos. Para finalizar la sesión, seleccione Detener sesión. Una vez finalizada la sesión, podrá obtener acceso al vídeo capturado y a los registros generados. Para obtener más información, consulte Obtener resultados de una sesión.

Sugerencias y trucos

En algunas regiones de AWS, puede experimentar problemas de desempeño con la sesión de acceso remoto. Esto se debe, en parte, a la latencia presente en algunas regiones. Si experimenta problemas de desempeño, dé una oportunidad a la sesión de acceso remoto para que se recupere antes de volver a interactuar con la aplicación.

Obtener resultados de una sesión de acceso remoto en AWS Device Farm

Para obtener información sobre sesiones, consulte Sesiones.

- Requisitos previos
- Visualización de detalles de una sesión
- Descarga de registros o vídeo de una sesión

Requisitos previos

- Complete una sesión. Siga las instrucciones de Usar una sesión de acceso remoto en AWS Device Farm y, a continuación, vuelva a esta página.

Visualización de detalles de una sesión

Cuando una sesión de acceso remoto finaliza, la consola de Device Farm muestra una tabla que contiene detalles sobre la actividad que tuvo lugar durante la sesión. Para obtener más información, consulte Análisis de información de registro.

Para volver a los detalles de una sesión en otro momento:

1. En el panel de navegación de Device Farm, seleccione Pruebas de dispositivos móviles y, a continuación, seleccione Proyectos.
2. Seleccione el proyecto que contiene la sesión.
3. Seleccione Acceso remoto y, a continuación, seleccione la sesión que desee revisar de la lista.

Descarga de registros o vídeo de una sesión

Cuando una sesión de acceso remoto finaliza, la consola de Device Farm proporciona acceso a una captura de vídeo de la sesión junto con los registros de actividad. En los resultados de la sesión, seleccione la pestaña Archivos para obtener una lista de enlaces a los registros y al vídeo de la sesión. Puede ver estos archivos en el navegador o guardarlos localmente.

Trabajar con dispositivos privados en AWS Device Farm

Un dispositivo privado es un dispositivo móvil físico que AWS Device Farm implementa en su nombre en un centro de datos de Amazon. Este dispositivo es exclusivo de tu AWS cuenta.

> **Note**
>
> Actualmente, los dispositivos privados solo están disponibles en la región AWS EE.UU. Oeste (Oregón) (`us-west-2`).

Si dispone de una flota de dispositivos privados, puede crear sesiones de acceso remoto y programar ejecuciones de prueba con sus dispositivos privados. También puede crear perfiles de instancia para controlar el comportamiento de los dispositivos privados durante una sesión de acceso remoto o una ejecución de prueba. Para obtener más información, consulte Administración de dispositivos privados en AWS Device Farm. Si lo desea, puede solicitar que determinados dispositivos privados Android se desplieguen como dispositivos rooteados.

También puede crear un servicio de punto de enlace de Amazon Virtual Private Cloud para probar aplicaciones privadas a las que su compañía tiene acceso, pero que no están disponibles a través de Internet. Por ejemplo, puede tener una aplicación web que se ejecute en la VPC que desea probar en dispositivos móviles. Para obtener más información, consulte Uso de los servicios de puntos de conexión de VPC de Amazon con Device Farm.

Si desea utilizar una flota de uno o varios dispositivos privados, contacte con nosotros. El equipo de Device Farm debe trabajar contigo para configurar e implementar una flota de dispositivos privados para tu AWS cuenta.

Temas

Administración de dispositivos privados en AWS Device Farm

Un dispositivo privado es un dispositivo móvil físico que AWS Device Farm implementa en su nombre en un centro de datos de Amazon. Este dispositivo es exclusivo de su cuenta de AWS.

> **Note**
>
> Los dispositivos privados actualmente solo están disponibles en la región Oeste de EE. UU. (Oregón)AWS (`us-west-2`).

Puede configurar una flota que contenga uno o varios dispositivos privados. Estos dispositivos se dedican a su cuenta de AWS. Después de configurar los dispositivos, puede crear uno o varios perfiles de instancia para ellos. Los perfiles de instancia puede ayudarle a automatizar las ejecuciones de prueba y a aplicar de forma coherente la misma configuración a las instancias de los dispositivos.

En este tema se explica cómo crear un perfil de instancia y cómo realizar otras tareas comunes de administración de dispositivos.

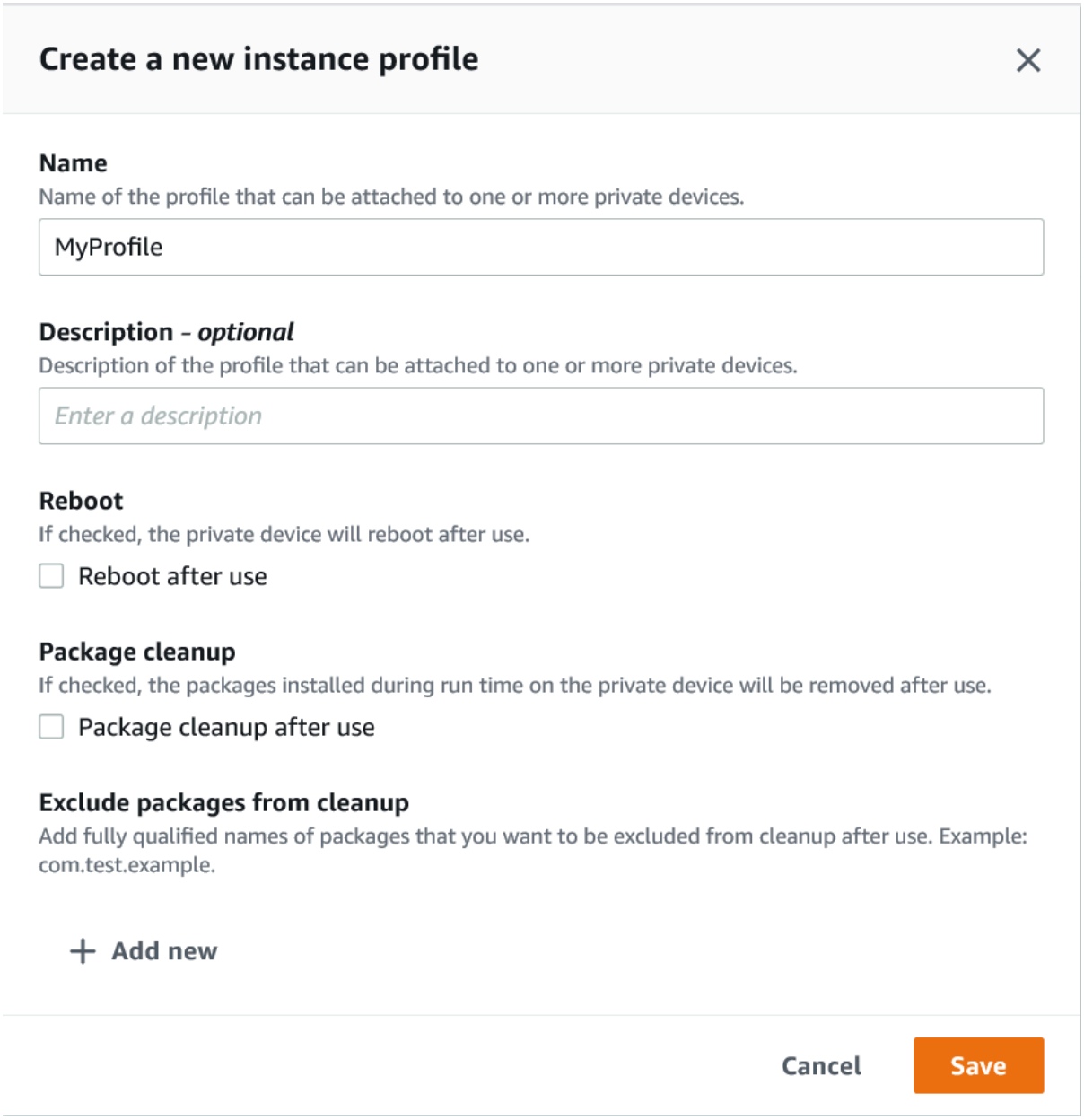
Temas

Creación de un perfil de instancia

Para controlar el comportamiento de los dispositivos privados durante una ejecución de prueba o una sesión de acceso remoto, puede crear o modificar un perfil de instancia en Device Farm. No se requieren perfiles de instancia para comenzar a utilizar los dispositivos privados.

1. Abra la consola Device Farm en https://console.aws.amazon.com/devicefarm/.
2. En el panel de navegación de Device Farm, seleccione Pruebas en dispositivos móviles y, a continuación, seleccione Dispositivos privados.
3. Seleccione Perfiles de instancia.
4. Seleccione Crear un perfil de instancia.

5. Introduzca un nombre para el perfil de instancia.



6. (Opcional) Escriba una descripción para el perfil de instancia.
7. (Opcional) Cambie cualquiera de los siguientes ajustes para especificar qué acciones desea que Device Farm realice en un dispositivo después de cada ejecución de prueba o finalización de sesión:

   - Reiniciar después del uso: para reiniciar el dispositivo, active esta casilla de verificación. De forma predeterminada, la casilla está desactivada (`false`).
   - Limpieza de paquetes: para conservar todos los paquetes de aplicaciones que ha instalado en el dispositivo, active esta casilla de verificación. De forma predeterminada, la casilla está

desactivada (`false`). Para conservar todos los paquetes de aplicaciones que ha instalado en el dispositivo, deje la casilla sin marcar.

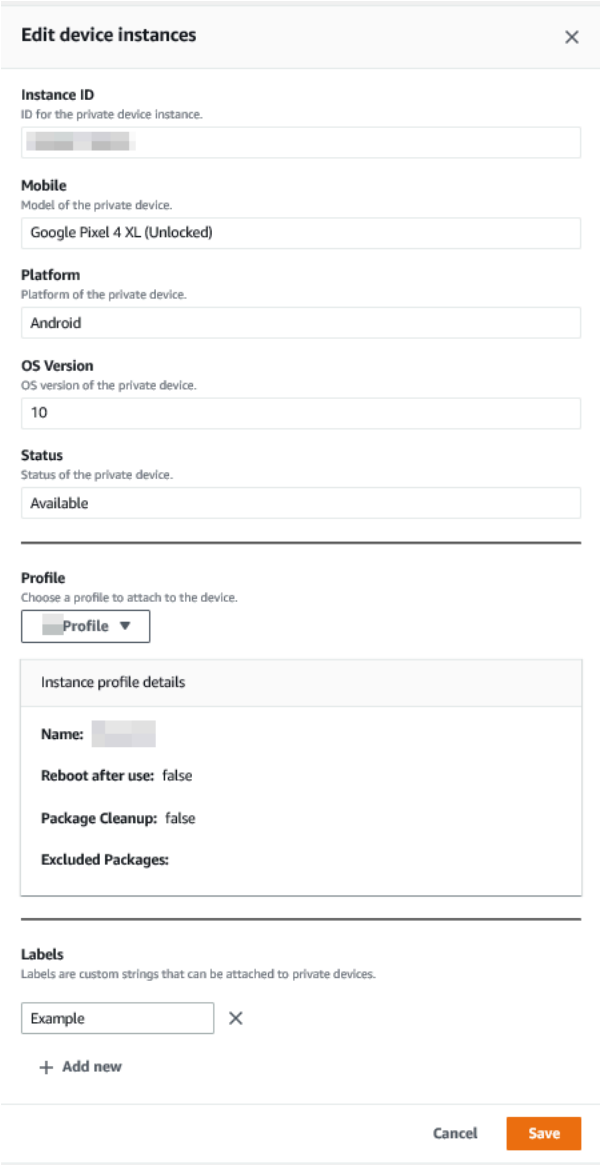
- Excluir paquetes de la limpieza: para conservar solo los paquetes de aplicaciones seleccionados en el dispositivo, seleccione la casilla Limpieza de paquetes y, a continuación, seleccione Agregar nuevo. Como nombre del paquete, especifique el nombre completo del paquete de aplicaciones que desea conservar en el dispositivo (por ejemplo, `com.test.example`). Para conservar más paquetes de aplicaciones en el dispositivo, seleccione Agregar nuevo y, a continuación, escriba el nombre completo de cada paquete.

8. Seleccione Save.

Administración de una instancia de dispositivo privado

Si ya tiene uno o varios dispositivos privados en su flota, puede ver información sobre cada instancia de dispositivo y administrar ciertos ajustes. También puede solicitar una instancia de dispositivo privado adicional.

1. Abra la consola Device Farm en https://console.aws.amazon.com/devicefarm/.
2. En el panel de navegación de Device Farm, seleccione Pruebas en dispositivos móviles y, a continuación, seleccione Dispositivos privados.
3. Seleccione Instancias de dispositivo. La pestaña Instancias de dispositivo muestra una tabla de los dispositivos privados que están en su flota. Para buscar en la tabla o filtrarla rápidamente, escriba los términos de búsqueda en los campos sobre las columnas.
4. (Opcional) Para solicitar una nueva instancia de dispositivo privado, seleccione Solicitar una nueva instancia de dispositivo o contacte con nosotros. Los dispositivos privados requieren configuración adicional con ayuda del equipo de Device Farm.
5. En la tabla de instancias de dispositivo, seleccione la opción situada junto a la instancia de la que desee ver información o gestionar y, a continuación, seleccione Editar.

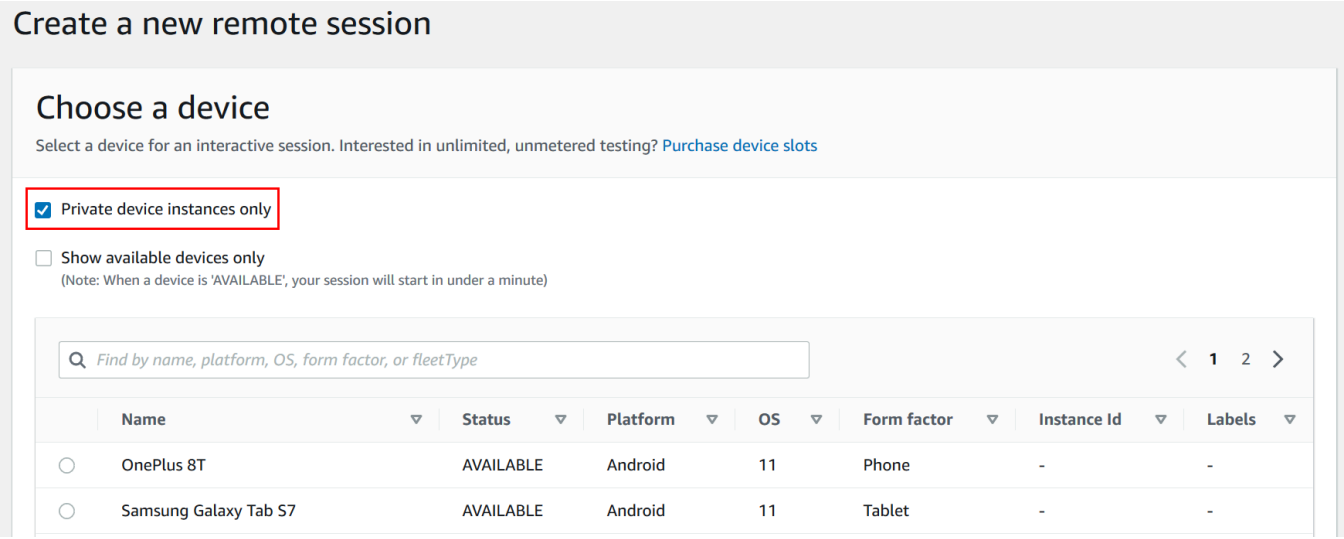


6. (Opcional) Para Perfil, seleccione un perfil de instancia para asociarlo a la instancia de dispositivo. Esto puede resultar útil si, por ejemplo, desea excluir siempre un paquete de aplicaciones específico de las tareas de limpieza.
7. (Opcional) En Etiquetas, seleccione Añadir nueva para añadir una etiqueta a la instancia de dispositivo. Las etiquetas pueden ayudarle a categorizar sus dispositivos y a encontrar dispositivos específicos con mayor facilidad.
8. Seleccione Save.

Creación de una ejecución de prueba o inicio de una sesión de acceso remoto

Después de configurar una flota de dispositivos privados, puede crear ejecuciones de prueba o iniciar sesiones de acceso remoto con uno o varios dispositivos privados de la flota.

1. Abra la consola Device Farm en https://console.aws.amazon.com/devicefarm/.
2. En el panel de navegación de Device Farm, seleccione Pruebas de dispositivos móviles y, a continuación, seleccione Proyectos.
3. Seleccione un proyecto existente de la lista o cree uno nuevo. Para crear un nuevo proyecto, seleccione Nuevo proyecto, indique un nombre para el proyecto y seleccione Enviar.
4. Haga una de las siguientes acciones:
   - Para crear una ejecución de prueba, seleccione Pruebas automatizadas y, a continuación, seleccione Crear una nueva ejecución. El asistente le guía a través de los pasos necesarios para crear la ejecución. En el paso Seleccionar dispositivos, puede editar un grupo de dispositivos existente o crear uno nuevo que incluya solo los dispositivos privados que el equipo de Device Farm haya configurado y asociado a su cuenta de AWS. Para obtener más información, consulte the section called “Crear un grupo de dispositivos privados”.
   - Para iniciar una sesión de acceso remoto, seleccione Acceso remoto y, a continuación, seleccione Iniciar una nueva sesión. En la página Elegir un dispositivo, seleccione Solo instancias de dispositivos privados para limitar la lista solo a los dispositivos privados que el equipo de Device Farm haya configurado y asociado a su cuenta de AWS. A continuación, seleccione el dispositivo al que desea obtener acceso, escriba un nombre para la sesión de acceso remoto y seleccione Confirmar e iniciar sesión.



Pasos siguientes

Después de configurar los dispositivos privados, también puede administrarlos de las siguientes maneras:

- Omitir la nueva firma de aplicación en dispositivos privados
- Utilice los servicios de puntos de conexión de Amazon Virtual Private Cloud con Device Farm

Para eliminar un perfil de instancia, en el menú Perfiles de instancia, seleccione la opción situada junto a la instancia que desee eliminar y, a continuación, seleccione Eliminar.

Selección de dispositivos privados en un grupo de dispositivos

Para usar dispositivos privados en la prueba, puede crear un grupo de dispositivos que seleccione sus dispositivos privados. Los grupos de dispositivos le permiten seleccionar dispositivos privados principalmente mediante tres tipos de reglas de grupos de dispositivos:

1. Reglas en función del ARN del dispositivo
2. Reglas en función de la etiqueta de una instancia de dispositivo
3. Reglas en función del ARN de una instancia

En las siguientes secciones, se describen en profundidad cada tipo de regla y sus casos de uso. Puede usar la consola Device Farm, la AWSinterfaz de la línea de comandos (AWSCLI) o la API

de Device Farm para crear o modificar un grupo de dispositivos con dispositivos privados mediante estas reglas.

Temas

ARN del dispositivo

El ARN de un dispositivo es un identificador que representa un tipo de dispositivo en lugar de una instancia específica de dispositivo físico. Un tipo de dispositivo se define mediante los siguientes atributos:

- El identificador de flota del dispositivo
- El fabricante original del dispositivo
- El número de modelo del dispositivo
- La versión del sistema operativo del dispositivo.
- El estado del dispositivo que indica si está rooteado o no

Muchas instancias de dispositivos físicos se pueden representar mediante un único tipo de dispositivo, donde cada instancia de ese tipo tiene los mismos valores para estos atributos. Por ejemplo, si tuviera tres dispositivos `Apple iPhone 13` con iOS `16.1.0` en su flota privada, cada dispositivo compartiría el mismo ARN del dispositivo. Si se agregara o eliminara algún dispositivo de su flota con estos mismos atributos, el ARN del dispositivo seguiría representando todos los dispositivos disponibles que tuviera en su flota para ese tipo de dispositivo.

El ARN de dispositivos es la forma más sólida de seleccionar dispositivos privados para un grupo de dispositivos, ya que permite que el grupo de dispositivos continúe seleccionando dispositivos independientemente de las instancias de dispositivos específicas que haya implementado en un momento dado. Las instancias de dispositivos privados individuales pueden sufrir fallos de hardware, lo que hace que Device Farm las sustituya automáticamente por nuevas instancias funcionales del

mismo tipo de dispositivo. En estos escenarios, la regla ARN del dispositivo garantiza que el grupo de dispositivos pueda seguir seleccionando dispositivos en caso de que se produzca un fallo de hardware.

Cuando utiliza una regla de ARN de dispositivo para los dispositivos privados de su grupo de dispositivos y programas una ejecución de prueba con ese grupo, Device Farm comprobará automáticamente qué instancias de dispositivos privados están representadas por el ARN de ese dispositivo. De las instancias que están disponibles actualmente, se asignará una de ellas para ejecutar la prueba. Si no hay ninguna instancia disponible actualmente, Device Farm esperará a que esté disponible la primera instancia disponible del ARN de ese dispositivo y la asignará para ejecutar la prueba.

Etiquetas de instancia de dispositivo

Una etiqueta de instancia de dispositivo es un identificador textual que se puede adjuntar como metadatos para una instancia de dispositivo. Puede adjuntar varias etiquetas a cada instancia de dispositivo y la misma etiqueta a varias instancias de dispositivo. Para obtener más información sobre cómo añadir, modificar o eliminar etiquetas de dispositivos de las instancias de dispositivos, consulte Administrar dispositivos privados.

La etiqueta de instancia de dispositivo puede ser una forma eficaz de seleccionar dispositivos privados para un grupo de dispositivos, ya que, si tiene varias instancias de dispositivos con la misma etiqueta, permite al grupo de dispositivos seleccionar cualquiera de ellos para la prueba. Si el ARN del dispositivo no es una buena regla para su caso de uso (por ejemplo, si desea seleccionar dispositivos de varios tipos de dispositivos o si desea seleccionar entre un subconjunto de todos los dispositivos de un tipo de dispositivo), las etiquetas de instancia de dispositivo pueden permitirle seleccionar varios dispositivos para su conjunto de dispositivos con mayor granularidad. Las instancias de dispositivos privados individuales pueden sufrir fallos de hardware, lo que hace que Device Farm las sustituya automáticamente por nuevas instancias funcionales del mismo tipo de dispositivo. En estos escenarios, la instancia del dispositivo de reemplazo no conservará ningún metadato de la etiqueta de instancia del dispositivo reemplazado. Por lo tanto, si aplica la misma etiqueta de instancia de dispositivo a varias instancias de dispositivo, la regla de etiqueta de instancia de dispositivo garantiza que su grupo de dispositivos pueda seguir seleccionando instancias de dispositivos en caso de que se produzca un fallo de hardware.

Cuando utiliza una regla de etiqueta de instancia de dispositivo para los dispositivos privados de su grupo de dispositivos y programas una ejecución de prueba con ese grupo, Device Farm comprobará automáticamente qué instancias de dispositivos privados están representadas por esa etiqueta de

instancia de dispositivo y, de esas instancias, seleccionará aleatoriamente una que esté disponible para ejecutar la prueba. Si no hay ninguna disponible, Device Farm seleccionará aleatoriamente cualquier instancia de dispositivo con la etiqueta de instancia de dispositivo para ejecutar la prueba y pondrá la prueba en cola para que se ejecute en el dispositivo cuando esté disponible.

ARN de instancia

El ARN de una instancia de dispositivo es un identificador que representa una instancia de dispositivo física básica tipo bare metal desplegada en una flota privada. Por ejemplo, si tuviera tres dispositivos `iPhone 13` con OS `15.0.0` en su flota privada y cada dispositivo compartiera el mismo ARN de dispositivo, cada dispositivo también tendría su propio ARN de instancia que representaría solo esa instancia.

El ARN de instancia de dispositivo es la forma menos sólida de seleccionar dispositivos privados para un grupo de dispositivos y solo se recomienda si los ARN de los dispositivos y las etiquetas de las instancias de dispositivos no se ajustan a su caso de uso. Los ARN de instancias de dispositivo suelen usarse como reglas para grupos de dispositivos cuando una instancia de dispositivo específica se configura de una manera única y específica como requisito previo para la prueba y si es necesario conocer y verificar esa configuración antes de ejecutar la prueba en ella. Las instancias de dispositivos privados individuales pueden sufrir fallos de hardware, lo que hace que Device Farm las sustituya automáticamente por nuevas instancias funcionales del mismo tipo de dispositivo. En estos escenarios, la instancia del dispositivo de reemplazo tendrá un ARN de instancia de dispositivo diferente al del dispositivo reemplazado. Por lo tanto, si depende de los ARN de instancias de dispositivos para su grupo de dispositivos, tendrá que cambiar manualmente la definición de reglas del grupo de dispositivos, pasando de usar el ARN anterior a usar el ARN nuevo. Si necesita preconfigurar manualmente el dispositivo para su prueba, este puede ser un flujo de trabajo eficaz (en comparación con los ARN de los dispositivos). Para realizar pruebas a escala, se recomienda intentar adaptar estos casos de uso para que funcionen con etiquetas de instancias de dispositivos y, si es posible, tener varias instancias de dispositivos preconfiguradas para las pruebas.

Cuando utiliza una regla de ARN de instancia de dispositivo para los dispositivos privados de su grupo de dispositivos y programas una ejecución de prueba con ese grupo, Device Farm asignará automáticamente esa prueba a esa instancia de dispositivo. Si esa instancia de dispositivo no está disponible, Device Farm pondrá en cola la prueba en el dispositivo cuando esté disponible.

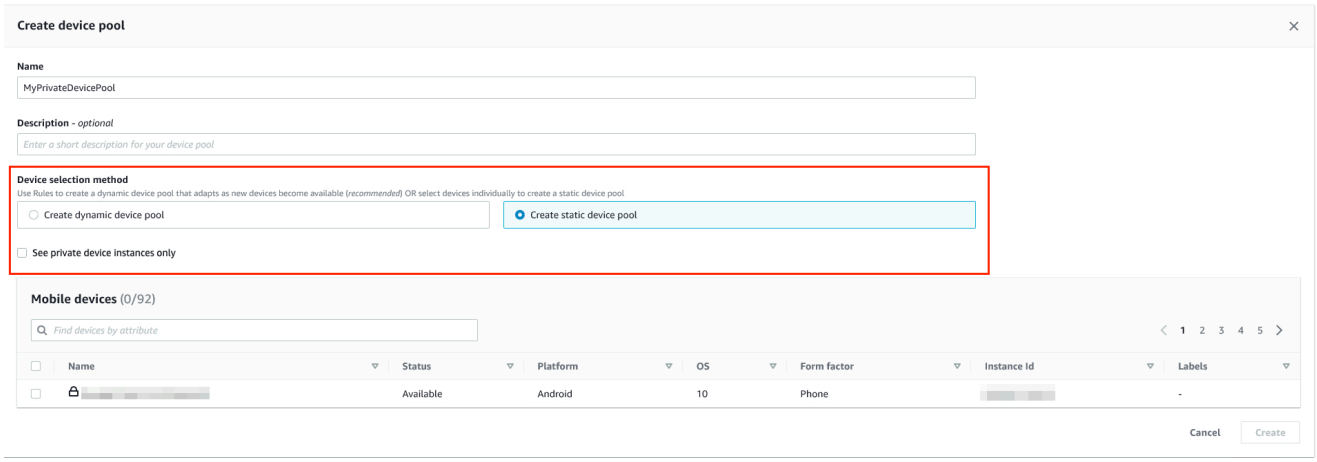
Crear un grupo de dispositivos privados con dispositivos privados (consola)

Al crear una ejecución de prueba, puede crear un grupo de dispositivos para esta y asegurarse de que el grupo solo incluye sus dispositivos privados.

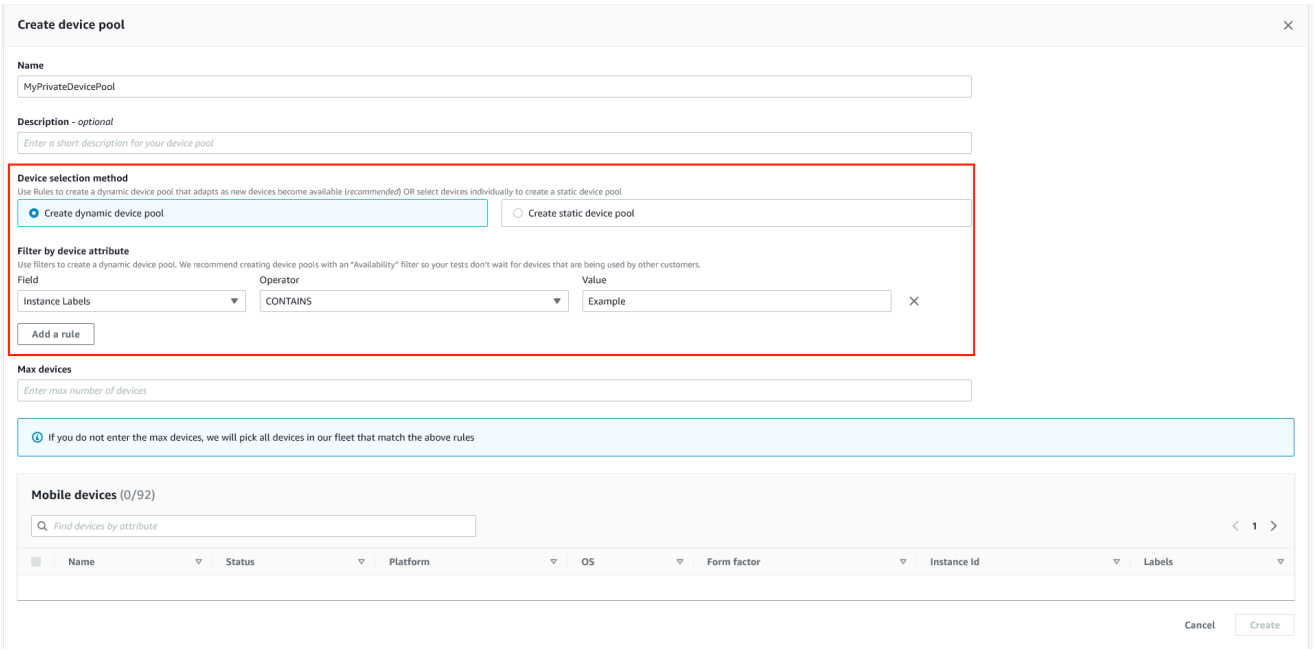
> **Note**
>
> Al crear un grupo de dispositivos con dispositivos privados en la consola, solo puede usar una de las tres reglas disponibles para seleccionar dispositivos privados. Si desea crear un grupo de dispositivos que contenga varios tipos de reglas para dispositivos privados (por ejemplo, grupos de dispositivos que contienen reglas para los ARN de los dispositivos y los ARN de las instancias de los dispositivos), debe crear el grupo mediante la CLI o la API.

1. Abra la consola Device Farm en https://console.aws.amazon.com/devicefarm/.
2. En el panel de navegación de Device Farm, seleccione Pruebas de dispositivos móviles y, a continuación, seleccione Proyectos.
3. Seleccione un proyecto existente de la lista o cree uno nuevo. Para crear un nuevo proyecto, seleccione Nuevo proyecto, indique un nombre para el proyecto y seleccione Enviar.
4. Seleccione Pruebas automatizadas y, a continuación, Crear una nueva ejecución. El asistente le guía a través de los pasos necesarios para elegir la aplicación y configurar la prueba que desea ejecutar.
5. En el paso Seleccionar dispositivos, seleccione Crear un nuevo grupo de dispositivos y escriba un nombre y una descripción opcional para el grupo de dispositivos.

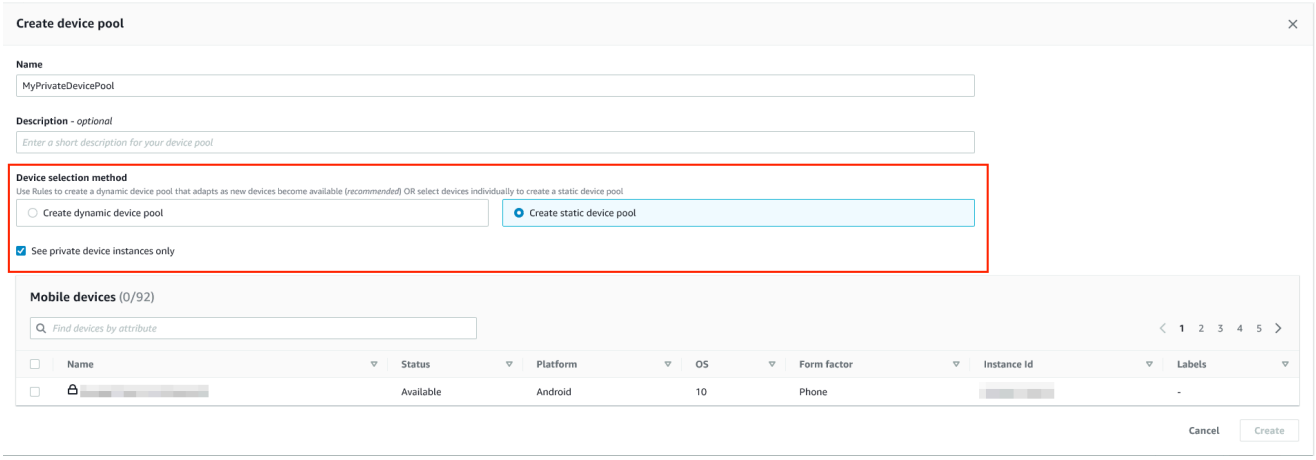
   a. Para usar las reglas de ARN de dispositivos para su grupo de dispositivos, seleccione Crear grupo de dispositivos estáticos y, a continuación, seleccione los tipos de dispositivos específicos de la lista que le gustaría usar en el grupo de dispositivos. No seleccione Solo instancias de dispositivos privados porque esta opción hace que el grupo de dispositivos se cree con reglas de ARN de instancia de dispositivo (en lugar de reglas de ARN de dispositivo).



b. Para usar las reglas de etiquetas de instancias de dispositivos para tu grupo de dispositivos, seleccione Crear grupo de dispositivos dinámico. A continuación, seleccione Agregar una regla para cada etiqueta que quiera usar en el grupo de dispositivos. Para cada regla, seleccione Etiquetas de instancia como `Field`, seleccione Contiene como `Operator` y especifique la etiqueta de instancia del dispositivo que desee como `Value`.



c. Para usar las reglas de ARN de instancias de dispositivos para su grupo de dispositivos, seleccione Crear grupo de dispositivos estático y, a continuación, seleccione Solo instancias de dispositivos privados para limitar la lista de dispositivos a las instancias de dispositivos privados que Device Farm tenga asociadas a su cuenta de AWS.



6. Seleccione Crear.

Crear un grupo de dispositivos privados con dispositivos privados (AWS CLI)

- Ejecute el comando .create-device-pool

Para obtener más información sobre el uso de Device Farm con la AWS CLI, consulte Referencia de AWS CLI.

Crear un grupo de dispositivos privados con dispositivos privados (API)

- Llame a la API `CreateDevicePool`.

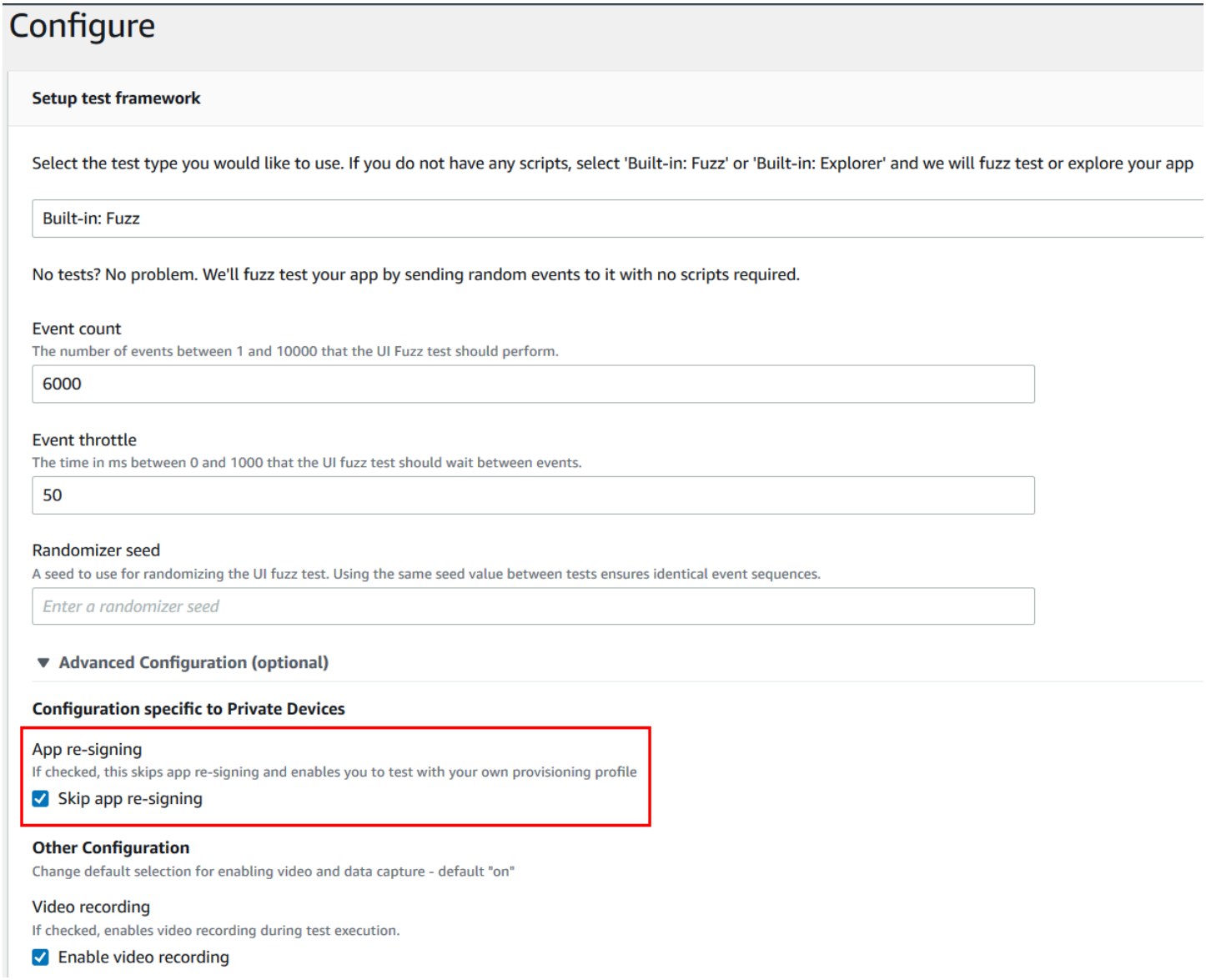
Para obtener más información acerca del uso de la API de Device Farm, consulte Automatización de Device Farm.

Omitir la nueva firma de aplicación en dispositivos privados en AWS Device Farm

Cuando utiliza dispositivos privados, puede omitir el paso en el que AWS Device Farm vuelve a firmar la aplicación. En los dispositivos públicos es distinto, pues en ellos Device Farm siempre vuelve a firmar la aplicación en las plataformas iOS y Android.

La nueva firma de la aplicación puede omitirse al crear una sesión de acceso remoto o una ejecución de prueba. Esto puede ser útil si la aplicación tiene cierta funcionalidad que se interrumpe cuando Device Farm la vuelve a firmar. Por ejemplo, es posible que las notificaciones de inserción no funcionen después de volver a firmar. Para obtener más información acerca de los cambios que Device Farm hace al probar la aplicación, consulte las preguntas frecuentes sobre AWS Device Farm.

Para omitir la nueva firma de la aplicación para una ejecución de prueba, seleccione Omitir volver a firmar aplicaciones en la página Configurar al crear la ejecución de prueba.



> **Note**
>
> Si utiliza el marco XCTest, la opción Omitir volver a firmar aplicaciones no está disponible. Para obtener más información, consulte Uso de XCTest para iOS y AWS Device Farm.

Los pasos adicionales para configurar la firma de aplicaciones varían, en función de si se utilizan dispositivos privados iOS o Android.

Omisión de la nueva firma de aplicación en dispositivos Android

Si va a probar la aplicación en un dispositivo Android privado, seleccione Omitir volver a firmar aplicaciones al crear la ejecución de prueba o la sesión de acceso remoto. No se necesitan más configuraciones.

Omisión de la nueva firma de aplicación en dispositivos iOS

Apple requiere que firme las aplicaciones para pruebas antes de que se puedan cargar en un dispositivo. En el caso de los dispositivos iOS, dispone de dos opciones para firmar la aplicación.

- Si utiliza un perfil de desarrollador interno (Enterprise), vaya directamente a la sección siguiente, the section called “Crear una sesión de acceso remoto para confiar en la aplicación”.

- Si utiliza un perfil de desarrollo de aplicaciones iOS ad hoc, primero debe registrar el dispositivo en su cuenta de desarrollador de Apple y, a continuación, actualizar su perfil de aprovisionamiento para incluir el dispositivo privado. A continuación, debe volver a firmar la aplicación con el perfil de aprovisionamiento que ha actualizado. Después, puede ejecutar la aplicación nuevamente firmada en Device Farm.

Para registrar un dispositivo con un perfil de aprovisionamiento de desarrollo de aplicaciones iOS (Ad-hoc)

1. Inicie sesión en su cuenta de desarrollador de Apple.
2. Vaya a la sección Certificados, ID y perfiles de la consola.
3. Vaya a Dispositivos.
4. Registre el dispositivo en la cuenta de desarrollador de Apple. Para obtener el nombre y el UDID del dispositivo, utilice la operación `ListDeviceInstances` de la API Device Farm.
5. Vaya a su perfil de aprovisionamiento y seleccione Editar.
6. Elija el dispositivo en la lista.
7. En Xcode, recupere el perfil de aprovisionamiento actualizado y, a continuación, vuelva a firmar la aplicación.

No se necesitan más configuraciones. Ahora puede crear una sesión de acceso remoto o una ejecución de prueba y seleccionar Omitir volver a firmar aplicaciones.

Creación de una sesión de acceso remoto para confiar en la aplicación de iOS

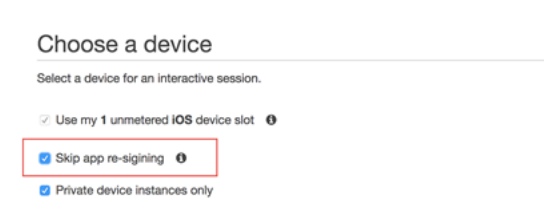
Si utiliza un perfil de aprovisionamiento de desarrollador interno (Enterprise), debe llevar a cabo un procedimiento único para confiar en el certificado de desarrollador de aplicaciones interno en cada uno de los dispositivos privados.

Para ello, puede instalar la aplicación que desea probar en el dispositivo privado o bien puede instalar una aplicación ficticia que esté firmada con el mismo certificado que la aplicación que desea probar. La instalación de una aplicación ficticia firmada con el mismo certificado tiene una ventaja. Una vez que se confía en el desarrollador del perfil de configuración o de aplicaciones empresariales, se confía en todas las aplicaciones de ese desarrollador en el dispositivo privado hasta que se eliminan. Por lo tanto, cuando cargue una nueva versión de la aplicación que desea probar, no tendrá que volver a confiar en el desarrollador. Esto resulta muy útil si ejecuta automatizaciones de prueba y no desea crear una sesión de acceso remoto cada vez que pruebe la aplicación.

Antes de iniciar la sesión de acceso remoto, siga los pasos en Creación de un perfil de instancia para crear o modificar un perfil de instancia en Device Farm. En el perfil de instancia, añada el ID de paquete de la aplicación de prueba o la aplicación ficticia a Excluir paquetes de la limpieza. A continuación, asocie este perfil de instancia a la instancia de dispositivo privado para asegurarse de que Device Farm no quita esta aplicación del dispositivo antes de iniciar una nueva ejecución de prueba. De este modo, se garantiza que su certificado de desarrollador sigue siendo de confianza.

Puede cargar la aplicación ficticia en el dispositivo mediante una sesión de acceso remoto, lo que le permite lanzar la aplicación y confiar en el desarrollador.

1. Siga las instrucciones de Crear una sesión para crear una sesión de acceso remoto que utilice el perfil de instancia de dispositivo privado que ha creado. Al crear la sesión, asegúrese de seleccionar Omitir volver a firmar aplicaciones.



> **Important**
>
> Para filtrar la lista de dispositivos de forma que solo incluya dispositivos privados, seleccione Solo instancias de dispositivos privados para asegurarse de que está utilizando un dispositivo privado con el perfil de instancia correcto.

Asegúrese también de añadir la aplicación ficticia o la aplicación que desea probar a Excluir paquetes de la limpieza en el perfil de instancia asociado a esta instancia.

2. Cuando se inicie la sesión remota, seleccione Elegir archivo para instalar una aplicación que utiliza su perfil de aprovisionamiento interno.
3. Lance la aplicación que acaba de cargar.
4. Siga las instrucciones para confiar en el certificado de desarrollador.

Todas las aplicaciones del desarrollador del perfil de configuración o de aplicaciones empresariales son ya de confianza en este dispositivo privado hasta que las elimine.

Uso de los servicios de puntos de conexión de VPC de Amazon con Device Farm

> **Note**
>
> El uso de puntos de conexión de VPC de Amazon con Device Farm solo es compatible con clientes con dispositivos privados configurados. Para habilitar su cuenta de AWS para utilizar esta característica con dispositivos privados, contacte con nosotros.

Amazon Virtual Private Cloud (Amazon VPC) es un servicio de AWS que puede utilizar para lanzar recursos de AWS en una red virtual (red privada virtual) que defina. Con una VPC, puede controlar la configuración de la red, como el rango de direcciones IP, las subredes, las tablas de enrutamiento y las puertas de enlace de red.

Si utiliza Amazon VPC como host para aplicaciones privadas en la costa Oeste de EE. UU. (Oregón) (`us-west-2`) AWS puede establecer una conexión privada entre su VPC y Device Farm. Con esta conexión, puede usar Device farm para probar aplicaciones privadas sin exponerlas a través de

Internet público. Para habilitar su cuenta de AWS a fin de utilizar esta característica con dispositivos privados, contacte con nosotros.

Para conectar un recurso de la VPC a Device Farm, puede utilizar la consola VPC de Amazon para crear un servicio de punto de conexión de VPC. Este servicio de punto de conexión le permite proporcionar el recurso de la VPC a Device Farm, través de un punto de conexión de VPC. El servicio de punto de conexión ofrece conectividad escalable de confianza con Device Farm sin necesidad de utilizar una puerta de enlace de Internet, una instancia de traducción de direcciones de red (NAT) o una conexión de VPN. Para obtener más información, consulte Servicios de punto de conexión de VPC (AWS PrivateLink) en la AWS PrivateLinkGuía.

> **Important**
>
> La característica de punto de conexión de VPC de Device Farm le ayuda a conectar de forma segura los servicios internos privados de su VPC a la VPC pública de Device Farm mediante conexiones de PrivateLink AWS. Aunque la conexión es segura y privada, la seguridad depende de la protección de sus credenciales de AWS. Si se vulneran sus credenciales de AWS, un atacante puede acceder a sus datos de servicio o exponerlos al público.

Después de crear un servicio de punto de conexión de VPC en Amazon VPC, puede utilizar la consola de Device Farm para crear un punto de conexión de VPC en Device Farm. En este tema se muestra cómo crear la conexión de Amazon VPS y la configuración de punto de conexión de VPC en Device Farm.

Antes de empezar

La siguiente información está destinada a los usuarios de Amazon VPC de la región Oeste de EE. UU. (Oregón) (`us-west-2`), con una subred en cada una de las siguientes zonas de disponibilidad: us-west-2a, us-west-2b y us-west-2c.

Device Farm tiene requisitos adicionales para los servicios de punto de conexión de VPC con los que se puede utilizar. Cuando cree y configure un servicio de punto de conexión de VPC que funcione con Device Farm, asegúrese de elegir opciones que cumplan los siguientes requisitos:

- Las zonas de disponibilidad del servicio deben incluir us-west-2a, us-west-2b y us-west-2c. El equilibrador de carga de red de asociado al servicio de punto de conexión de VPC determina las zonas de disponibilidad para un servicio de punto de conexión de VPC. Si el servicio de punto de conexión de VPC no muestra estas tres zonas de disponibilidad, debe volver a crear el equilibrador

de carga de red para habilitarlas y, a continuación, volver a asociar dicho equilibrador de carga de red al servicio de punto de conexión.

- Las entidades principales permitidas en el servicio de punto de conexión deben incluir el nombre de recurso de Amazon (ARN) del punto de conexión de VPC de Device Farm (ARN del servicio). Después de crear el servicio de punto de conexión, añada el ARN del servicio de punto de conexión de VPC de Device Farm a la lista blanca para otorgar a Device Farm permiso de acceso a dicho servicio. Para obtener el ARN del servicio de punto de conexión de VPC de Device Farm, contacte con nosotros.

Además, si mantiene activada la configuración Aceptación obligatoria al crear el servicio de punto de conexión de VPC, debe aceptar manualmente cada solicitud de conexión que Device Farm envíe al servicio de punto de conexión. Para cambiarla, seleccione el servicio de punto de conexión en la consola de VPC de Amazon, seleccione Acciones y, a continuación, seleccione Modificar la configuración de aceptación de punto de enlace. Para obtener más información, consulte Cambiar los equilibradores de carga y la configuración de aceptación en la AWS PrivateLink Guía.

En la siguiente sección se explica cómo crear un servicio de punto de conexión de VPC que cumpla estos requisitos.

Paso 1: Creación de un equilibrador de carga de red

El primer paso para establecer una conexión privada entre la VPC y Device Farm consiste en crear un equilibrador de carga de red para enrutar las solicitudes a un grupo objetivo.

New console

Si desea crear un equilibrador de carga de red con la consola nueva

1. Abra la consola de Amazon Elastic Compute Cloud (Amazon EC2) en https://console.aws.amazon.com/ec2/.
2. En el panel de navegación, en Equilibrio de carga, seleccione Equilibradores de carga.
3. Elija Crear un equilibrador de carga.
4. En Equilibrador de carga de red, seleccione Crear.
5. En la página Crear un equilibrador de carga de red, en Configuración básica, haga lo siguiente:

    a. Introduzca un nombre para el equilibrador de carga.

b. En Esquema, seleccione Interno.

6. En Mapeo de red, realice lo siguiente:

   a. Seleccione la VPC para su grupo objetivo.

   b. Seleccione las siguientes Asignaciones:

      - `us-west-2a`
      - `us-west-2b`
      - `us-west-2c`

7. En Agentes de escucha y direccionamiento, utilice las opciones de Protocolo y Puerto para elegir su grupo objetivo.

   > **Note**
   >
   > El equilibrador de carga entre zonas de disponibilidad está deshabilitado de forma predeterminada.
   > Como el equilibrador de carga usa las zonas de disponibilidad `us-west-2a`, `us-west-2b` y `us-west-2c`, requiere que los objetivos estén registrados en cada una de esas zonas de disponibilidad o, si registra los objetivos en menos de las tres zonas, requiere que habilite el equilibrador de carga entre zonas. De lo contrario, es posible que el equilibrador de carga no funcione según lo esperado.

8. Elija Crear un equilibrador de carga.

Old console

Si desea crear un equilibrador de carga de red con la consola anterior

1. Abra la consola de Amazon Elastic Compute Cloud (Amazon EC2) en https://console.aws.amazon.com/ec2/.
2. En el panel de navegación, en Equilibrio de carga, seleccione Equilibradores de carga.
3. Elija Crear un equilibrador de carga.
4. En Equilibrador de carga de red, seleccione Crear.
5. En la página Configurar balanceador de carga, en Configuración básica, haga lo siguiente:

   a. Introduzca un nombre para el equilibrador de carga.

b. En Esquema, seleccione Interno.

6. En Agentes de escucha, seleccione el Protocolo y el Puerto que utiliza su grupo objetivo.
7. En Zonas de disponibilidad, haga lo siguiente:

   a. Seleccione la VPC para su grupo objetivo.

   b. Seleccione las siguientes Zonas de disponibilidad:

      - `us-west-2a`
      - `us-west-2b`
      - `us-west-2c`

   c. Seleccione Siguiente: Configurar los ajustes de seguridad.
8. (Opcional) Configure los ajustes de seguridad y, a continuación, seleccione Siguiente: Configurar el enrutamiento.
9. En la página Configuración del enrutamiento, haga lo siguiente:

   a. En Target group, elija Existing target group.

   b. En Nombre, seleccione su grupo objetivo.

   c. Seleccione Siguiente: Registrar destinos.
10. En la página Registrar destinos, revise sus objetivos y, a continuación, seleccione Siguiente: Revisión.

> **Note**
>
> El equilibrador de carga entre zonas de disponibilidad está deshabilitado de forma predeterminada.
> Como el equilibrador de carga usa las zonas de disponibilidad `us-west-2a`, `us-west-2b` y `us-west-2c`, requiere que los objetivos estén registrados en cada una de esas zonas de disponibilidad o, si registra los objetivos en menos de las tres zonas, requiere que habilite el equilibrador de carga entre zonas. De lo contrario, es posible que el equilibrador de carga no funcione según lo esperado.

11. Revise la configuración de su equilibrador de carga y seleccione Crear.

Paso 2: Crear un servicio de punto de conexión de VPC

Tras crear el equilibrador de carga de red, utilice la consola de Amazon VPC para crear un servicio de punto de conexión en su VPC.

1. Abra la consola de Amazon VPC en https://console.aws.amazon.com/vpc/.
2. En Recursos por región, seleccione Servicios de punto de conexión.
3. Seleccione Crear servicio de punto de conexión.
4. Haga una de las siguientes acciones:
   - Si ya tiene un equilibrador de carga de red y desea que el servicio de punto de conexión lo utilice, selecciónelo en Balanceadores de carga disponibles y vaya al paso 5.
   - Si aún no ha creado un equilibrador de carga de red, seleccione Crear nuevo equilibrador de carga. Se abre la consola de Amazon EC2. Siga los pasos de Creación de un equilibrador de carga de red empezando por el paso 3 y, a continuación, continúe con estos pasos en la consola de Amazon VPC.
5. En Zonas de disponibilidad incluidas, compruebe que `us-west-2a`, `us-west-2b` y `us-west-2c` aparecen en la lista.
6. Si no desea tener que aceptar o denegar manualmente cada una de las solicitudes de conexión que se envíe al servicio de punto de conexión, en Configuración adicional, desactive Aceptación obligatoria. Si desactiva esta casilla, el servicio de punto de enlace acepta automáticamente cada solicitud de conexión que recibe.
7. Seleccione Crear.
8. En servicio de punto de conexión, seleccione Permitir entidades principales.
9. Contacte con nosotros para obtener el ARN del punto de conexión de VPC de Device Farm para añadirlo a la lista blanca para el servicio de punto de conexión y, después, añada dicho ARN del servicio a la lista blanca del servicio.
10. En la pestaña Detalles del servicio de punto de conexión, anote el nombre del servicio (nombre de servicio). Necesitará este nombre al crear la configuración del punto de enlace de la VPC en el siguiente paso.

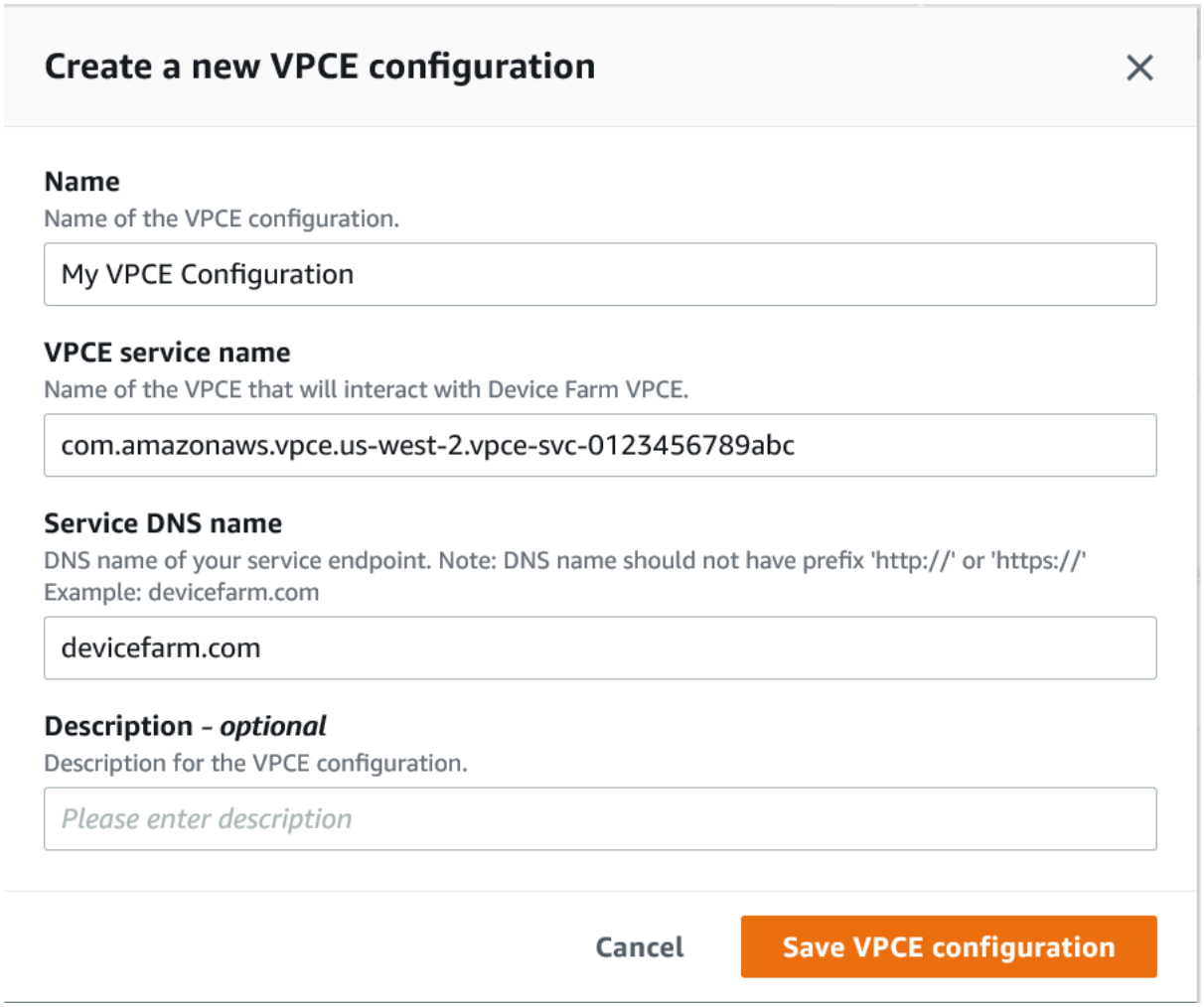
Su servicio de punto de conexión de VPC ya está listo para utilizarlo con Device Farm.

Paso 3: Crear una configuración de punto de conexión de VPC en Device Farm

Después de crear un servicio de punto de conexión en Amazon VPC, puede crear una configuración de punto de conexión de VPC en Device Farm.

1. Inicie sesión en la consola de Device Farm en https://console.aws.amazon.com/devicefarm.
2. En el panel de navegación, seleccione Pruebas de dispositivos móviles y, a continuación, Dispositivos privados.
3. Seleccione Configuraciones de VPCE.
4. Seleccione Crear una configuración de VPCE.
5. En Crear una nueva configuración de VPCE, introduzca un Nombre para la configuración del punto de conexión de VPC.
6. En Nombre del servicio VPCE, escriba el nombre del servicio de punto de conexión de VPC (nombre de servicio) que anotó en la consola de Amazon VPC. El nombre se parece a `com.amazonaws.vpce.us-west-2.vpce-svc-id`.
7. En Nombre del servicio DNS, escriba el nombre de DNS de servicio para la aplicación que desea probar (por ejemplo, `devicefarm.com`). No especifique `http` o `https` antes del nombre de DNS de servicio.

   El nombre de dominio no está accesible a través de Internet público. Además, este nuevo nombre de dominio, que se mapea a su servicio de punto de conexión de VPC, lo genera Amazon Route 53 y está disponible solo para usted en su sesión de Device Farm.
8. Seleccione Save.



Paso 4: Crear una ejecución de prueba

Una vez guardada la configuración de punto de conexión de VPC, puede utilizar la configuración para crear ejecuciones de prueba o sesiones de acceso remoto. Para obtener más información, consulte Crear una ejecución de prueba en Device Farm o Crear una sesión.

Trabajo con Amazon VPC entre AWS regiones

Los servicios de Device Farm solo se encuentran en la región Oeste de EE. UU. (Oregón`us-west-2`) (). Puede usar Amazon Virtual Private Cloud (Amazon VPC) para acceder a un servicio de su Amazon Virtual Private Cloud de otra AWS región mediante Device Farm. Si Device Farm y su servicio se encuentran en la misma región, consulte Uso de los servicios de puntos de conexión de VPC de Amazon con Device Farm.

Hay dos formas de acceder a sus servicios privados ubicados en una región diferente. Si tiene servicios ubicados en otra región que no es `us-west-2`, puede usar la vinculación de VPC para vincular la VPC de esa región a otra VPC que esté interactuando con Device Farm en `us-west-2`. Sin embargo, si tiene servicios en varias regiones, una puerta de enlace de tránsito le permitirá acceder a esos servicios con una configuración de red más sencilla.

Para obtener más información, consulte Escenarios de interconexión de VPC en la Guía de interconexión de Amazon VPC.

Emparejamiento de VPC

Puede conectar dos VPC situadas en regiones distintas, siempre y cuando tengan bloques de CIDR diferenciados que no se solapen. De este modo, se garantiza que todas las direcciones IP privadas sean únicas y se permite que todos los recursos de las VPC se comuniquen entre sí sin necesidad de traducción de direcciones de red (NAT). Para obtener más información acerca de la notación CIDR, consulte RFC 4632.

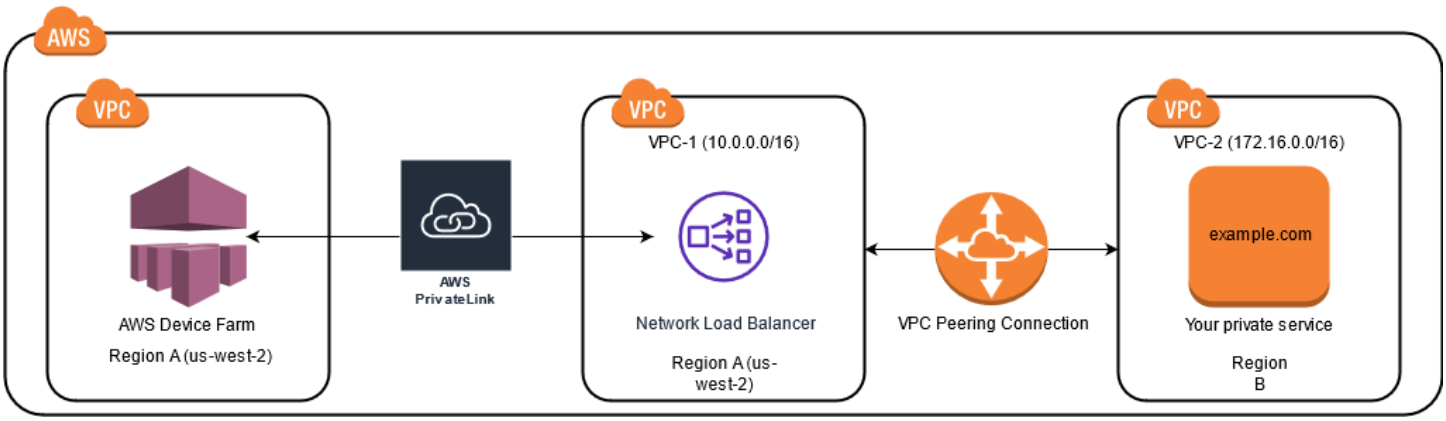
En este tema se incluye un escenario de ejemplo entre regiones en el que Device Farm (denominado VPC-1) se encuentra en la región de Oeste de EE. UU. (Oregón) (`us-west-2`). La segunda VPC del ejemplo se encuentra en otra región y se denomina VPC-2.

Ejemplo de VPC entre regiones en Device Farm

Componente de VPC	VPC-1	VPC-2
CIDR	10.0.0.0/16	172.16.0.0/16

> **Important**
>
> El establecimiento de una conexión de vinculación entre dos VPC puede cambiar la postura de seguridad de las VPC. Además, añadir nuevas entradas a sus tablas de enrutamiento puede cambiar la postura de seguridad de los recursos de las VPC. Es su responsabilidad implementar estas configuraciones de manera que cumplan con los requisitos de seguridad de su organización. Para más información, consulte el Modelo de responsabilidad compartida.

En el siguiente diagrama se muestran los componentes del ejemplo y las interacciones entre ellos.



Temas

Requisitos previos

Este ejemplo requiere lo siguiente:

- Dos VPC configuradas con subredes que contienen bloques CIDR que no se superponen.
- La VPC-1 debe estar en la región `us-west-2` y contener subredes para las zonas de disponibilidad `us-west-2a`, `us-west-2b` y `us-west-2c`.

Para obtener información acerca de la creación de VPC y configuración de subredes, consulte Uso de VPC y subredes en la Guía de interconexión de Amazon VPC.

Paso 1: Configurar una conexión de emparejamiento entre VPC-1 y VPC-2

Establezca una conexión de emparejamiento entre las dos VPC que contengan bloques CIDR que no se superpongan. Para ello, consulte Crear y aceptar conexiones de emparejamiento de VPC en la Guía de emparejamiento de VPC de Amazon. Con el escenario entre regiones de este tema y la Guía de emparejamiento de VPC de Amazon, se crea el siguiente ejemplo de configuración de conexión de emparejamiento:

Nombre

```
Device-Farm-Peering-Connection-1
```

ID de VPC (solicitante)

```
vpc-0987654321gfedcba (VPC-2)
```

Cuenta

```
My account
```

Región

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

ID de VPC (aceptador)

```
vpc-1234567890abcdefg (VPC-1)
```

> **Note**
>
> Asegúrese de consultar las cuotas de conexión de emparejamiento de VPC al establecer nuevas conexiones de emparejamiento. Para obtener más información, consulte Cuotas de VPC de Amazon en la Guía de emparejamiento de VPC de Amazon

Paso 2: Actualizar las tablas de enrutamiento en VPC-1 y VPC-2

Tras configurar una conexión de emparejamiento, debe establecer una ruta de destino entre las dos VPC para transferir datos entre ellas. Para establecer esta ruta, puede actualizar manualmente la tabla de enrutamiento de la VPC-1 para que apunte a la subred de la VPC-2 y viceversa. Para ello, consulte Actualizar las tablas de enrutamiento para una conexión de emparejamiento de VPC en la

Guía de conexión de emparejamiento de VPC de Amazon. Con el escenario entre regiones de este tema y la Guía de emparejamiento de VPC de Amazon, se crea el siguiente ejemplo de configuración de tabla de enrutamiento:

Ejemplo de tabla de enrutamiento de VPC

Componente de VPC	VPC-1	VPC-2
ID de tabla de ruteo	rtb-1234567890abcdefg	rtb-0987654321gfedcba
Rango de direcciones locales	10.0.0.0/16	172.16.0.0/16
Rango de direcciones de destino	172.16.0.0/16	10.0.0.0/16

Paso 3: Crear un grupo de destino

Después de configurar las rutas de destino, puede configurar un equilibrador de carga de red en la VPC-1 para enrutar las solicitudes a la VPC-2.

El equilibrador de carga de red debe incluir primero un grupo objetivo que contenga las direcciones IP a las que se envían las solicitudes.

Creación de un grupo de destino

1. Identifique las direcciones IP del servicio al que quiere dirigirse en la VPC-2.
   - Estas direcciones IP deben ser miembros de la subred utilizada en la conexión de emparejamiento.
   - Las direcciones IP de destino deben ser estáticas e inmutables. Si su servicio tiene direcciones IP dinámicas, considere la posibilidad de dirigirse a un recurso estático (como un equilibrador de carga de red) y hacer que ese recurso estático dirija las solicitudes a su verdadero objetivo.

     > **Note**
     >
     > - Si se dirige a una o más instancias independientes de Amazon Elastic Compute Cloud (Amazon EC2), abra la consola de Amazon EC2 en https://console.aws.amazon.com/ec2/ y, a continuación, seleccione Instances.
     > - Si se dirige a un grupo de instancias de Amazon EC2 Auto Scaling, debe asociar el grupo Auto Scaling de Amazon EC2 a un equilibrador de carga de red. Para obtener

más información, consulte Adjuntar un equilibrador de carga al grupo de escalado automático en la Guía del usuario de Amazon EC2 Auto Scaling.

A continuación, puede abrir la consola Amazon EC2 en https://console.aws.amazon.com/ec2/ y, a continuación, seleccionar Interfaces de red. Desde allí, puede ver las direcciones IP de cada una de las interfaces de red del equilibrador de carga de red en cada Zona de disponibilidad.

2. Cree un grupo objetivo en la VPC-1. Para obtener más información, consulte Crear grupos de destino para equilibradores de carga de red en la Guía del usuario para equilibradores de carga de red.

   Los grupos objetivo de los servicios de una VPC diferente requieren la siguiente configuración:

   - En Elegir un tipo de destino, elija Direcciones IP.
   - Para la VPC, elija la VPC host del equilibrador de carga. Para el ejemplo del tema, será VPC-1.
   - En la página Registrar destinos, registre un destino para cada dirección IP de la VPC-2.

     En Red, seleccione Otra dirección IP privada.

     En Zona de disponibilidad, elija las zonas que desee en la VPC-1.

     Para la dirección IPv4 , elija la dirección IP de la VPC-2.

     En el caso de Puertos, elija los suyos.
   - Seleccione Incluir como pendiente debajo. Cuando haya terminado de especificar direcciones, seleccione Registrar destinos pendientes.

En el escenario interregional de este tema y en la Guía del usuario de los equilibradores de carga de red, se utilizan los siguientes valores en la configuración del grupo objetivo:

Tipo de objetivo

```
IP addresses
```

Tipo de grupo de destino

```
my-target-group
```

Protocolo/puerto

```
TCP : 80
```

VPC

```
vpc-1234567890abcdefg (VPC-1)
```

Red

```
Other private IP address
```

Zona de disponibilidad

```
all
```

Dirección IPv4

```
172.16.100.60
```

Puertos

```
80
```

Paso 4: Crear un nuevo equilibrador de carga de red

Cree un equilibrador de carga de red con el grupo objetivo descrito en el paso 3. Para ello, consulte Creación de un equilibrador de carga de red.

En el escenario entre regiones de este tema, se utilizan los siguientes valores en un ejemplo de configuración de equilibrador de carga de red:

Nombre del equilibrador de carga

```
my-nlb
```

Esquema

```
Internal
```

VPC

```
vpc-1234567890abcdefg (VPC-1)
```

Mapeo

```
us-west-2a - subnet-4i23iuufkdiufsloi
```

```
us-west-2b - subnet-7x989pkjj78nmn23j
```

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

Protocolo/puerto

```
TCP : 80
```

Grupo de destinos

```
my-target-group
```

Paso 5: Crear un servicio de punto de conexión de VPC

Puede crear un servicio de punto de conexión de VPC mediante el equilibrador de carga de red. A través de este servicio de punto de conexión de VPC, Device Farm puede conectarse a su servicio en la VPC-2 sin necesidad de ninguna infraestructura adicional, como una puerta de enlace de Internet, una instancia NAT o una conexión VPN.

Para ello, consulte Creación de un servicio de punto de conexión de VPC de Amazon.

Paso 6: Crear una configuración de punto de conexión de VPC en Device Farm

Ahora puede establecer una conexión privada entre su VPC y Device Farm. Puede usar Device Farm para probar servicios privados sin exponerlos a través de la Internet pública. Para ello, consulte Creación de una configuración de punto de conexión de VPC en Device Farm.

En el escenario entre regiones de este tema, se utilizan los siguientes valores en un ejemplo de configuración de punto de conexión de VPC:

Nombre

```
My VPCE Configuration
```

Nombre del servicio de VPCE

```
com.amazonaws.vpce.us-west-2.vpce-svc-1234567890abcdefg
```

Nombre del DNS del servicio

```
devicefarm.com
```

Paso 7: Crear una ejecución de prueba

Puede crear ejecuciones de prueba que utilicen la configuración de punto de conexión de VPC descrita en el paso 6. Para obtener más información, consulte Crear una ejecución de prueba en Device Farm o Crear una sesión.

Cree una red escalable con una puerta de enlace de tránsito

Para crear una red escalable con más de dos VPC, puede usar la puerta de enlace de tránsito como un centro de tránsito de red para interconectar las redes de las VPC y en las instalaciones. Para configurar una VPC en la misma región que Device Farm para usar una puerta de enlace de tránsito, puede seguir la guía Servicios de puntos de conexión de VPC de Amazon con Device Farm para segmentar los recursos de otra región en función de sus direcciones IP privadas.

Para obtener más información acerca de las puertas de enlace de tránsito, consulte Qué es una puerta de enlace de tránsito en Puertas de enlace de tránsito de VPC de Amazon.

Cerrar los dispositivos privados

> **Important**
>
> Estas instrucciones solo se aplican a la rescisión de los acuerdos de dispositivos privados. Para cualquier otro problema relacionado con AWS los servicios y la facturación, consulta la documentación correspondiente a esos productos o ponte en contacto con el servicio de AWS asistencia.

Para cancelar un dispositivo privado una vez transcurrido el plazo inicial acordado, debes avisar con 30 días de antelación si no se renueva a través de nuestro correo electrónico `<aws-devicefarm-support@amazon>` .com.

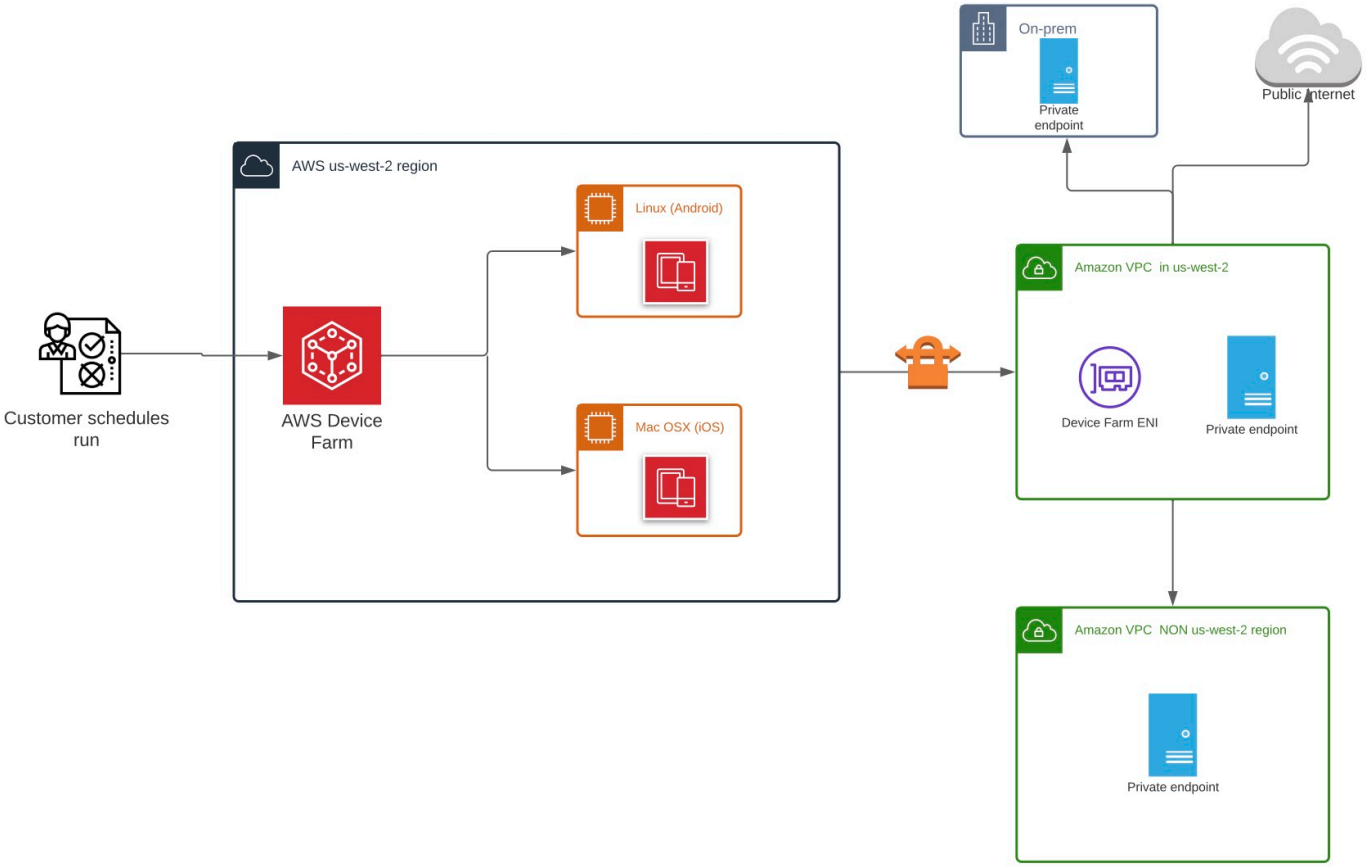
VPC-ENI en AWS Device Farm

> **Warning**
>
> Esta característica solo está disponible en dispositivos privados. Para solicitar el uso privado de un dispositivo en su cuenta AWS, contacte con nosotros. Si ya tiene dispositivos privados agregados a su cuenta AWS, le recomendamos encarecidamente que utilice este método de conectividad de VPC.

La característica de conectividad VPC-ENI de AWS Device Farm ayuda a los clientes a conectarse de forma segura a sus puntos de conexión privados con host en AWS, un software en las instalaciones, o a otro proveedor en la nube.

Puede conectar los dispositivos móviles de Device Farm y sus máquinas host a un entorno de Amazon Virtual Private Cloud (Amazon VPC) en la región `us-west-2`, lo que permite el acceso a servicios y aplicaciones aislados que no están conectados a Internet a través de una interfaz de red elástica. Para obtener más información sobre VPC, consulte la Guía del usuario de Amazon VPC.

Si su punto de conexión privado o VPC no se encuentra en la región `us-west-2`, puede vincularlo con una VPC de la región `us-west-2` mediante soluciones como una puerta de enlace de tránsito o Interconexión con VPC. En tales situaciones, Device Farm creará un ENI en una subred que usted proporcione para la VPC de su región `us-west-2` y usted será responsable de garantizar que se pueda establecer una conexión entre la VPC de la región `us-west-2` y la VPC de la otra región.



Para obtener información sobre cómo usar AWS CloudFormation para crear y vincular automáticamente las VPC, consulte las plantillas de interconexión de VPC en el AWS CloudFormation repositorio de plantillas de GitHub.

> **Note**
>
> Device Farm no cobra nada por la creación de ENI en la VPC de un cliente en `us-west-2`. El coste de la conectividad entre VPC externa o entre regiones no está incluido en esta característica.

Una vez que configure el acceso a la VPC, los dispositivos y las máquinas host que utilice para las pruebas no podrán conectarse a recursos ajenos a la VPC (por ejemplo, las CDN públicas) a menos

que haya una puerta de enlace NAT que especifique dentro de la VPC. Para obtener información, consulte Gateways NAT en la Guía del usuario de Amazon VPC.

Temas

Control de acceso a AWS e IAM

AWS Device Farm le permite usar AWS Identity and Access Management(IAM) para crear políticas que concedan o restrinjan el acceso a las funciones de Device Farm. Para utilizar la característica de conectividad de VPC con AWS Device Farm, se requiere la siguiente política de IAM para la cuenta de usuario o el rol que utilice para acceder a AWS Device Farm:

```
{
  "Version": "2012-10-17",
  "Statement": [{
      "Effect": "Allow",
      "Action": [
        "devicefarm:*",
        "ec2:DescribeVpcs",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:CreateNetworkInterface"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
      "Resource": "arn:aws:iam::*:role/aws-service-role/devicefarm.amazonaws.com/
AWSServiceRoleForDeviceFarm",
      "Condition": {
        "StringLike": {
```

```
          "iam:AWSServiceName": "devicefarm.amazonaws.com"
        }
      }
    }
  ]
}
```

Para crear o actualizar un proyecto de Device Farm con una configuración de VPC, su política de IAM debe permitirle realizar las siguientes acciones contra los recursos enumerados en la configuración de VPC:

```
"ec2:DescribeVpcs"
"ec2:DescribeSubnets"
"ec2:DescribeSecurityGroups"
"ec2:CreateNetworkInterface"
```

Además, su política de IAM también debe permitir la creación del rol vinculado al servicio:

```
"iam:CreateServiceLinkedRole"
```

> **Note**
>
> Ninguno de estos permisos es necesario para los usuarios que no utilizan configuraciones de VPC en sus proyectos.

Roles vinculados a servicios

AWS Device Farm utiliza roles vinculados a servicios AWS Identity and Access Management (IAM). Un rol vinculado a un servicio es un tipo único de rol de IAM que está vinculado directamente a Device Farm. Los roles vinculados a servicios están predefinidos por Device Farm e incluyen todos los permisos que el servicio requiere para llamar a otros servicios de AWS en su nombre.

Un rol vinculado a un servicio simplifica la configuración de Device Farm porque ya no tendrá que agregar manualmente los permisos necesarios. Device Farm define los permisos de sus roles vinculados a servicios y, a menos que esté definido de otra manera, solo Device Farm puede asumir sus roles. Los permisos definidos incluyen las políticas de confianza y de permisos y que la política de permisos no se pueda adjuntar a ninguna otra entidad de IAM.

Solo puede eliminar una función vinculada a un servicio después de eliminar sus recursos relacionados. De esta forma, se protegen los recursos de Device Farm, ya que se evita que se puedan eliminar accidentalmente permisos de acceso a los recursos.

Para obtener información sobre otros servicios que admiten roles vinculados a servicios, consulte Servicios de AWS que funcionan con IAM y busque los servicios que tienen Sí en la columna Rol vinculado a servicios. Seleccione una opción Sí con un enlace para ver la documentación acerca del rol vinculado al servicio en cuestión.

Permisos de roles vinculados a servicios de Device Farm

Device Farm usa el rol vinculado al servicio denominado AWSServiceRoleForDeviceFarm, que permite a Device Farm acceder a los recursos de AWS en su nombre.

El rol vinculado al servicio AWSServiceRoleForDeviceFarm depende de los siguientes servicios para asumir el rol:

- `devicefarm.amazonaws.com`

La política de permisos de rol permite que Device Farm realice las siguientes acciones:

- Para la cuenta
  - Crea interfaces de red
  - Describir las interfaces de red
  - Describir VPC
  - Describir subredes
  - Describir grupos de seguridad
  - Eliminar interfaces
  - Modificar interfaces de red
- Para interfaces de red
  - Crear etiquetas
- Para interfaces de red de EC2 administradas por Device Farm
  - Crear permisos de interfaz de red

La política de IAM completa dice lo siguiente:

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

Debe configurar permisos para permitir a una entidad de IAM (como un usuario, grupo o rol) crear, editar o eliminar un rol vinculado a servicios. Para obtener más información, consulte Permisos de roles vinculados a servicios en la Guía del usuario de IAM.

Creación de un rol vinculado a un servicio de Device Farm

Cuando proporciona una configuración de VPC para un proyecto de pruebas móviles, no necesita crear manualmente roles vinculados a servicios. Cuando se crea el primer recurso de Device Farm en la AWS Management Console, la AWS CLI, la o la API de AWS, Device Farm crea el rol vinculado al servicio automáticamente.

Si elimina este rol vinculado al servicio y necesita crearlo de nuevo, puede utilizar el mismo proceso para volver a crear el rol en su cuenta. Cuando se crea el primer recurso de Device Farm, Device Farm crea de nuevo el rol vinculado al servicio automáticamente.

También puede utilizar la consola de IAM para crear un rol vinculado al servicio con el caso de uso de Device Farm. En la AWS CLI o la API de AWS, cree un rol vinculado al servicio con el nombre de servicio `devicefarm.amazonaws.com`. Para obtener más información, consulte Crear un rol vinculado a un servicio en la Guía del usuario de IAM. Si elimina este rol vinculado al servicio, puede utilizar este mismo proceso para volver a crear el rol.

Modificación de un rol vinculado a un servicio de Device Farm

Device Farm no le permite modificar el rol vinculado al servicio AWSServiceRoleForDeviceFarm. Después de crear un rol vinculado a servicios, no puede cambiarle el nombre, ya que varias entidades pueden hacer referencia al mismo. Sin embargo, puede editar la descripción del rol mediante IAM. Para obtener más información, consulte Editar un rol vinculado a servicios en la Guía del usuario de IAM..

Eliminación de un rol vinculado a un servicio de Device Farm

Si ya no necesita utilizar una característica o servicio que requiere un rol vinculado a un servicio, recomendamos que elimine dicho rol. De esta forma no tiene una entidad no utilizada que no se monitorice ni mantenga de forma activa. Sin embargo, debe limpiar los recursos del rol vinculado al servicio antes de eliminarlo manualmente.

> **Note**
>
> Si el servicio de Device Farm está utilizando el rol cuando intenta eliminar los recursos, la eliminación podría producir un error. En tal caso, espere unos minutos e intente de nuevo la operación.

Para eliminar manualmente el rol vinculado a servicios mediante IAM

Utilice la consola de IAM, la AWS CLI o la API de AWS para eliminar el rol vinculado al servicio AWSServiceRoleForDeviceFarm. Para obtener más información, consulte Eliminación de un rol vinculado a servicios en la Guía del usuario de IAM.

Regiones admitidas para los roles vinculados a servicios de Device Farm

Device Farm admite el uso de roles vinculados a servicios en todas las regiones en las que el servicio está disponible. Para obtener más información, consulte Regiones y puntos de enlace de AWS.

Device Farm no admite el uso de roles vinculados a servicios en todas las regiones en las que el servicio está disponible. Puede utilizar el rol AWSServiceRoleForDeviceFarm en las regiones que se detallan a continuación.

Nombre de la región de	Identidad de la región	Compatibilidad en Device Farm
US East (N. Virginia)	us-east-1	No
US East (Ohio)	us-east-2	No
EE. UU. Oeste (Norte de California)	us-west-1	No
EE. UU. Oeste (Oregon)	us-west-2	Sí
Asia Pacífico (Mumbai)	ap-south-1	No
Asia Pacífico (Osaka)	ap-northeast-3	No
Asia Pacífico (Seúl)	ap-northeast-2	No

Nombre de la región de	Identidad de la región	Compatibilidad en Device Farm
Asia Pacífico (Singapur)	ap-southeast-1	No
Asia Pacífico (Sídney)	ap-southeast-2	No
Asia Pacífico (Tokio)	ap-northeast-1	No
Canadá (Central)	ca-central-1	No
Europe (Frankfurt)	eu-central-1	No
Europe (Ireland)	eu-west-1	No
Europe (London)	eu-west-2	No
Europa (París)	eu-west-3	No
América del Sur (São Paulo)	sa-east-1	No
AWS GovCloud (US)	us-gov-oeste-1	No

Requisitos previos

La siguiente lista describe algunos requisitos y sugerencias que se deben revisar al crear configuraciones de VPC-ENI:

- Los dispositivos privados deben estar asignados a su cuenta de AWS.
- Debe tener un usuario o un rol de cuenta de AWS con permisos para crear un rol vinculado al servicio. Cuando se utilizan puntos de enlace de Amazon VPC con funciones de pruebas móviles de Device Farm, Device Farm crea un rol vinculado al servicio AWS Identity and Access Management (IAM).
- Device Farm solo se puede conectar a las VPC de la región `us-west-2`. Si no dispone de una VPC en la región `us-west-2`, debe crear una. A continuación, para acceder a los recursos de una VPC de otra región, debe establecer una conexión de interconexión entre la VPC de la región `us-west-2` y la VPC de la otra región. Para obtener información sobre las interconexiones de VPC, consulte la Guía de interconexión de Amazon VPC.

Debe comprobar que tiene acceso a la VPC especificada al configurar la conexión. Debe configurar determinados permisos de Amazon Elastic Compute Cloud (Amazon EC2) para Device Farm.

- La resolución de DNS es necesaria en la VPC que utilice.
- Una vez creada la VPC, necesitará la siguiente información sobre la VPC de la región `us-west-2`:
  - VPC ID
  - ID de subred
  - ID de grupo de seguridad
- Debe configurar las conexiones de Amazon VPC por proyecto. En este momento, solo puede configurar una VPC por proyecto. Al configurar una VPC, Amazon VPC crea una interfaz dentro de la VPC y la asigna a las subredes y grupos de seguridad especificados. Todas las sesiones futuras asociadas al proyecto utilizarán la conexión de VPC configurada.
- No puede utilizar las configuraciones de VPC-ENI junto con la característica VPCE antigua.
- Recomendamos encarecidamente no actualizar un proyecto existente con una configuración de VPC-ENI, ya que los proyectos existentes pueden tener configuraciones de VPCE que persistan en el nivel de ejecución. En su lugar, si ya utiliza las funciones de VPCE existentes, utilice VPC-ENI para todos los proyectos nuevos.

Conexión con Amazon VPC

Puede configurar y actualizar su proyecto para utilizar puntos de conexión de VPC de Amazon. La configuración de la VPC-ENI se configura para cada proyecto. Un proyecto solo puede tener un punto de conexión VPC-ENI en un momento dado. Para configurar el acceso a la VPC para un proyecto, debe conocer los siguientes detalles:

- El ID de VPC en `us-west-2` si la aplicación está alojada allí o el ID de VPC de `us-west-2` que se conecta a otra VPC de una región diferente.
- Los grupos de seguridad aplicables que se van a aplicar a la conexión.
- Las subredes que se asociarán a la conexión. Cuando se inicia una sesión, se utiliza la subred más grande disponible. Recomendamos tener varias subredes asociadas a distintas zonas de disponibilidad para mejorar la situación de disponibilidad de la conectividad de la VPC.

Una vez que haya creado la configuración de VPC-ENI, puede actualizar sus detalles mediante la consola o la CLI siguiendo los pasos que se indican a continuación.

Console

1. Inicie sesión en la consola de Device Farm en https://console.aws.amazon.com/devicefarm.
2. En el panel de navegación de Device Farm, seleccione Pruebas de dispositivos móviles y, a continuación, seleccione Proyectos.
3. En Proyectos de pruebas en móviles, seleccione el nombre de su proyecto de la lista.
4. Elija Project settings (Configuración del proyecto).
5. En la sección Configuración de nube privada virtual (VPC), puede cambiar `VPC`, `Subnets` y `Security Groups`.
6. Seleccione Guardar.

CLI

Utilice los siguientes comandos de la CLI de AWS para actualizar la VPC de Amazon:

```
$  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
```

También puede configurar una Amazon VPC al crear su proyecto:

```
$  aws devicefarm create-project \
--name VPCDemo \
--vpc-config \
securityGroupIds=sg-02c1537701a7e3763,sg-005dadf9311efda25,\
subnetIds=subnet-09b1a45f9cac53717,subnet-09b1a45f9cac12345,\
vpcId=vpc-0238fb322af81a368
```

Límites

Las siguientes limitaciones se aplican a la característica VPC-ENI:

- Puede proporcionar hasta cinco grupos de seguridad en la configuración de VPC de un proyecto de Device Farm.
- Puede proporcionar hasta ocho subredes en la configuración de VPC de un proyecto de Device Farm.
- Al configurar un proyecto de Device Farm para que funcione con su VPC, la subred más pequeña que pueda proporcionar debe tener un mínimo de cinco direcciones IPv4 disponibles.
- Las direcciones IP públicas no son compatibles en este momento. En lugar de ello, le recomendamos que utilice subredes privadas en sus proyectos de Device Farm. Si necesita acceso público a Internet durante las pruebas, utilice una puerta de enlace de traducción de direcciones de red (NAT). La configuración de un proyecto de Device Farm con una subred pública no proporciona a las pruebas acceso a Internet ni una dirección IP pública.
- Solo se admite el tráfico saliente del ENI gestionado por el servicio. Esto significa que el ENI no puede recibir solicitudes entrantes no solicitadas de la VPC.

Registro de llamadas a la API de AWS Device Farm con AWS CloudTrail

AWS Device Farm se integra a AWS CloudTrail, un servicio que brinda un registro de las acciones que realiza un usuario, un rol o un servicio de AWS en AWS Device Farm. CloudTrail captura las llamadas a la API de AWS Device Farm como eventos. Las llamadas capturadas incluyen las llamadas desde la consola de AWS Device Farm y las llamadas desde el código a las operaciones de la API de AWS Device Farm. Si crea un registro de seguimiento, puede habilitar la entrega continua de eventos de CloudTrail a un bucket de Amazon S3, incluidos los eventos para AWS Device Farm. Si no configura un registro de seguimiento, puede ver los eventos más recientes de la consola de CloudTrail en el Historial de eventos. Mediante la información que recopila CloudTrail, puede determinar la solicitud que se realizó a AWS Device Farm, la dirección IP desde la que se realizó, quién la realizó y cuándo, entre otros detalles.

Para obtener más información acerca de CloudTrail, consulte la Guía del usuario de AWS CloudTrail.

Información de AWS Device Farm en CloudTrail

CloudTrail se habilita en su cuenta de AWS cuando la crea. Cuando se produce una actividad en AWS Device Farm, dicha actividad se registra en un evento de CloudTrail junto con los eventos de los demás servicios de AWS en Historial de eventos. Puede ver, buscar y descargar los últimos eventos de la cuenta de AWS. Para obtener más información, consulte Ver eventos con el historial de eventos de CloudTrail.

Para mantener un registro continuo de eventos en la cuenta de AWS, incluidos los eventos de AWS Device Farm, cree un registro de seguimiento. Un registro de seguimiento permite a CloudTrail enviar archivos de registro a un bucket de Amazon S3. De forma predeterminada, cuando se crea un registro de seguimiento en la consola, el registro de seguimiento se aplica a todas las regiones de AWS. El registro de seguimiento registra los eventos de todas las regiones de la partición de AWS y envía los archivos de registro al bucket de Amazon S3 especificado. También es posible configurar otros servicios de AWS para analizar en profundidad y actuar en función de los datos de eventos recopilados en los registros de CloudTrail. Para obtener más información, consulte los siguientes temas:

- Introducción a la creación de registros de seguimiento
- Servicios e integraciones compatibles con CloudTrail

- Configuración de notificaciones de Amazon SNS para CloudTrail
- Recibir archivos de registro de CloudTrail de varias regiones y Recibir archivos de registro de CloudTrail de varias cuentas

Cuando el registro de CloudTrail está habilitado en su cuenta de AWS, las llamadas a la API realizadas a acciones de Device Farm se registran en archivos de registro. Los registros de Device Farm se crean junto con los registros de otros servicios de AWS en un archivo de registro. CloudTrail determina cuándo debe crearse un nuevo archivo y escribir en él en función del periodo de tiempo y del tamaño del archivo.

Todas las acciones de Device Farm se registran y documentan en el Referencia de AWS CLI y el Automatización de Device Farm. Por ejemplo, las llamadas para crear un proyecto o una ejecución en Device Farm generan entradas en archivos de registro de CloudTrail.

Cada entrada de registro o evento contiene información sobre quién generó la solicitud. La información de identidad del usuario le ayuda a determinar lo siguiente:

- Si la solicitud se realizó con credenciales de usuario AWS Identity and Access Management (IAM) o credenciales de usuario raíz.
- Si la solicitud se realizó con credenciales de seguridad temporales de un rol o fue un usuario federado.
- Si la solicitud la realizó otro servicio de AWS.

Para obtener más información, consulte el Elemento userIdentity de CloudTrail.

Comprender las entradas de los archivos de registro de AWS Device Farm

Un registro de seguimiento es una configuración que permite la entrega de eventos como archivos de registros en un bucket de Amazon S3 que especifique. Los archivos log de CloudTrail pueden contener una o varias entradas de log. Un evento representa una solicitud específica realizada desde un origen y contiene información sobre la acción solicitada, la fecha y la hora de la acción, los parámetros de la solicitud, etc. Los archivos de registro de CloudTrail no rastrean el orden en la pila de las llamadas públicas a la API, por lo que estas no aparecen en ningún orden específico.

En el ejemplo siguiente, se muestra una entrada de registro de CloudTrail que ilustra la acción `ListRuns` de Device Farm:

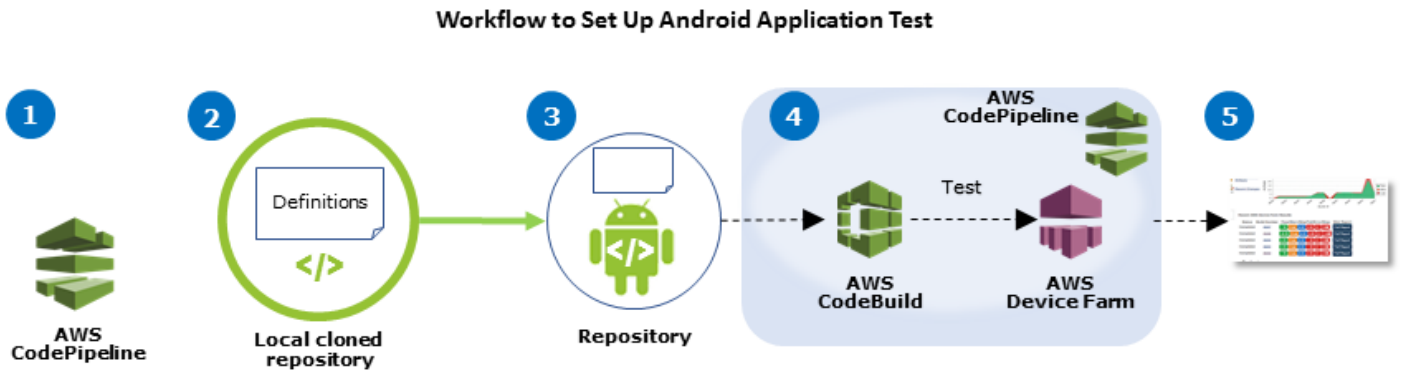
```
{
  "Records": [
    {
      "eventVersion": "1.03",
      "userIdentity": {
        "type": "Root",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:root",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext":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

Uso de AWS Device Farm en una etapa de prueba de CodePipeline

Puede utilizar AWS CodePipeline para incorporar pruebas de aplicaciones móviles configuradas en Device Farm en una canalización de publicación automatizada administrada por AWS. Puede configurar la canalización para ejecutar pruebas bajo demanda, de forma programada o como parte de un flujo de integración continua.

En el siguiente diagrama se muestra el flujo de integración continua en el que se crea y se prueba una aplicación Android cada vez que se envía una inserción a su repositorio. Para crear esta configuración de canalización, consulte Tutorial: Crear una canalización que compila y prueba su aplicación Android cuando se envía una confirmación al repositorio de GitHub.



1. Configuración	2. Añadir definiciones	3. Inserción	4. Compilar y probar	5. Informe
Configurar recursos de canalización	Añadir definiciones de compilación y prueba al paquete	Enviar un paquete al repositorio	Compilación y prueba de aplicación del artefacto de salida de compilación activado automáticamente	Ver resultados de la prueba

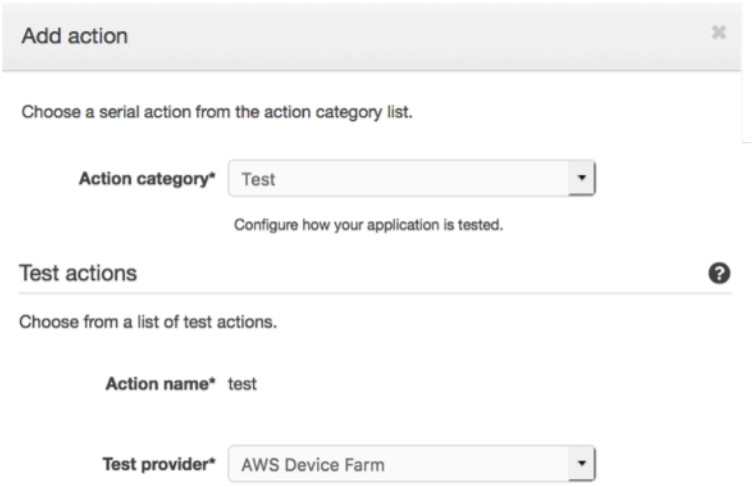
Para obtener información sobre cómo configurar una canalización que prueba continuamente una aplicación compilada (como un archivo `.ipa` de iOS o `.apk` de Android) como su origen, consulte Tutorial: Crear una canalización que compile y pruebe la aplicación iOS después de un cambio en el bucket de Amazon S3.

Configure CodePipeline para usar sus pruebas de Device Farm

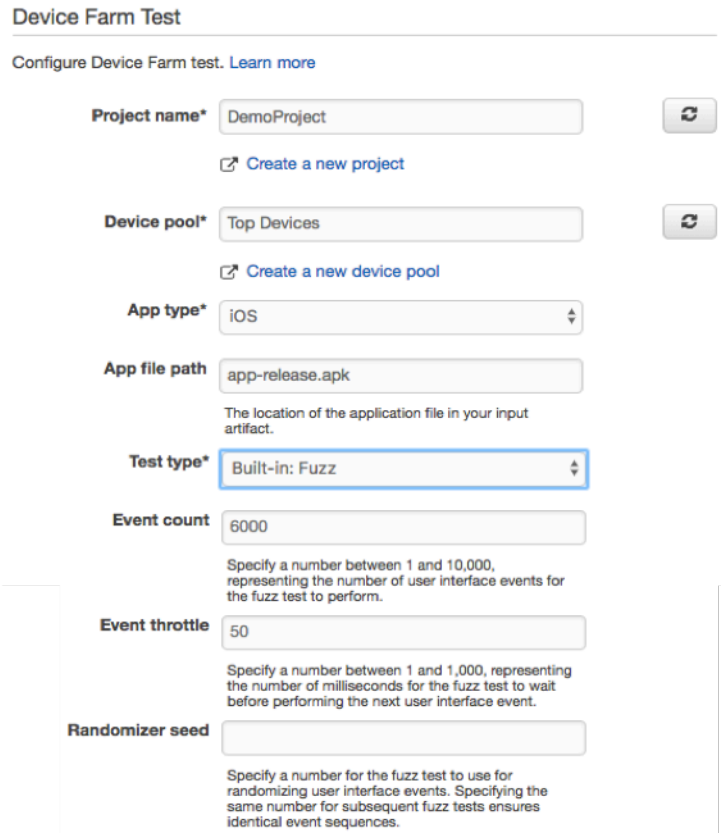
En estos pasos, se da por hecho que ha configurado un proyecto de Device Farm y ha creado una canalización. La canalización debe configurarse con una etapa de prueba que reciba un artefacto de entrada que contenga la definición de la prueba y los archivos de paquete de aplicación compilados. El artefacto de entrada de la etapa de prueba puede ser el artefacto de salida de una etapa de código fuente o de compilación configurada en la canalización.

Para configurar una ejecución de prueba de Device Farm como una acción de prueba de CodePipeline

1. Inicie sesión en la AWS Management Console y abra la consola de CodePipeline en https://console.aws.amazon.com/codepipeline/.
2. Elija la canalización para la publicación de su aplicación.
3. En el panel de la etapa de prueba, seleccione el icono del lápiz y, a continuación, seleccione Acción.
4. En el panel Añadir acción, en Categoría de acción, seleccione Probar.
5. En Nombre de la acción, escriba un nombre.
6. En Proveedor de la prueba, seleccione AWS Device Farm.

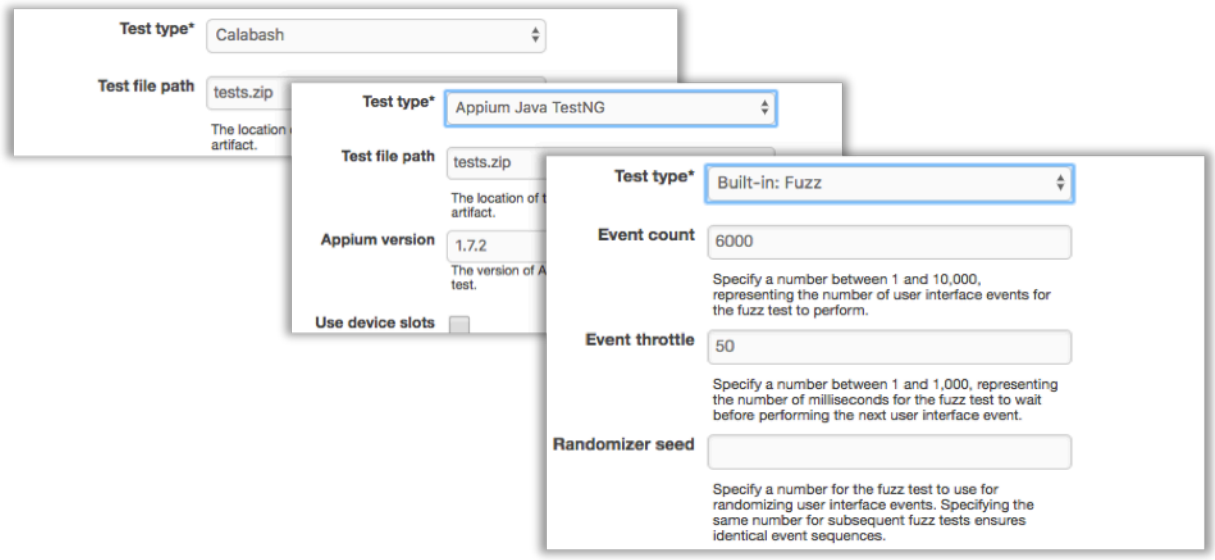


7. En Nombre del proyecto, seleccione su proyecto de Device Farm existente o seleccione Crear un nuevo proyecto.
8. En Grupo de dispositivos, seleccione el grupo de dispositivos existente o Crear un nuevo grupo de dispositivos. Si crea un grupo de dispositivos, debe seleccionar un conjunto de dispositivos de prueba.
9. En Tipo de aplicación, seleccione la plataforma de su aplicación.

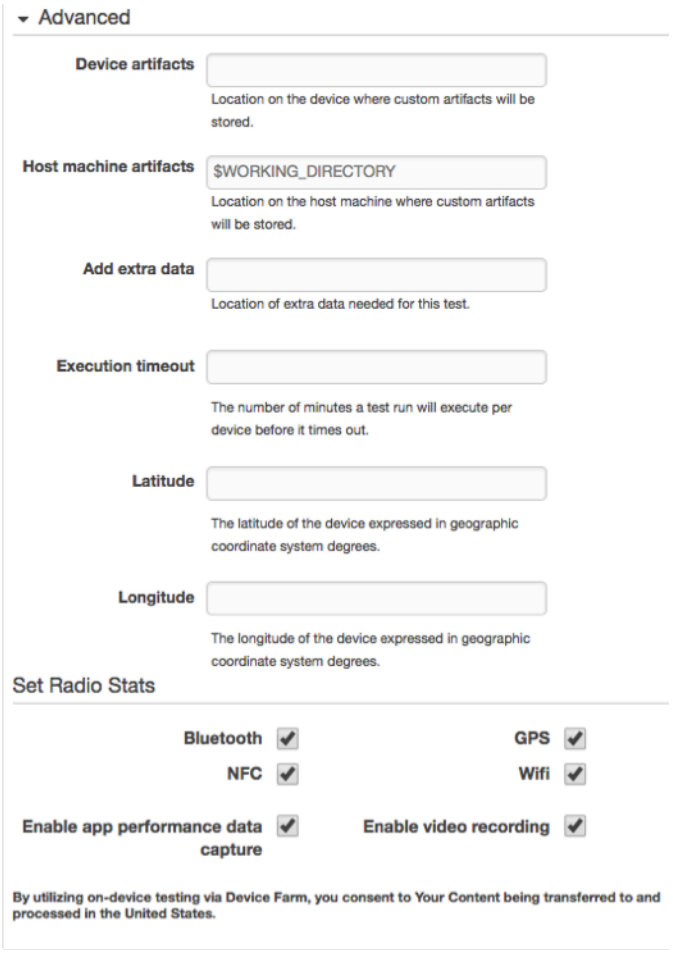


10. En Ruta de archivo de aplicación, escriba la ruta del paquete de aplicación compilado. La ruta es relativa a la raíz del artefacto de entrada de la prueba.
11. En Tipo de prueba, realice alguna de las siguientes operaciones:

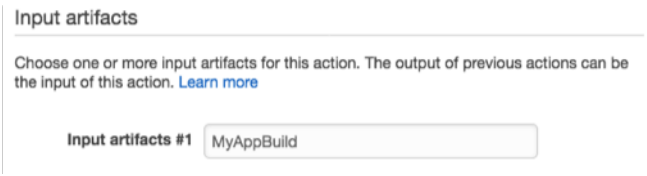
    - Si utiliza una de las pruebas de Device Farm integradas, elija el tipo de la prueba configurada en su proyecto de Device Farm.
    - Si no utiliza una de las pruebas integradas de Device Farm, en Ruta de archivo de prueba escriba la ruta del archivo de definición de prueba. La ruta es relativa a la raíz del artefacto de entrada de la prueba.



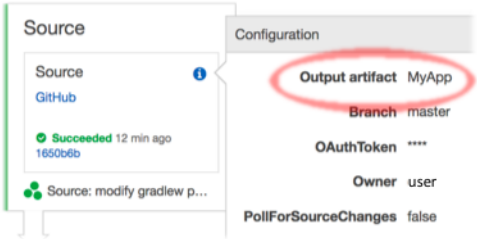
12. En los campos restantes, proporcione la configuración que sea adecuada para su prueba y tipo de aplicación.
13. (Opcional) En Avanzado, proporcione una configuración detallada de la ejecución de prueba.



14. En Artefactos de entrada, seleccione el artefacto de entrada que coincida con el artefacto de salida de la etapa anterior a la de prueba en la canalización.



En la consola de CodePipeline, puede encontrar el nombre del artefacto de salida de cada etapa si pasa el ratón sobre el icono de información en el diagrama de canalizaciones. Si su canalización prueba la aplicación directamente desde la fase Fuente, seleccione MyApp. Si su canalización incluye una etapa Compilación, seleccione MyAppBuild.



15. En la parte inferior del panel, seleccione Añadir acción.
16. En el panel de CodePipeline, seleccione Guardar los cambios en la canalización y, a continuación, Guardar cambios.
17. Para enviar los cambios y comenzar una compilación de canalización, seleccione Publicar modificación y, a continuación, Publicar.

AWS CLI referencia de AWS Device Farm

Para usar AWS Command Line Interface (AWS CLI) para ejecutar los comandos de Device Farm, consulte la AWS CLIReferencia de AWS Device Farm.

Para obtener información general sobre AWS CLI, consulte la Guía del usuario de AWS Command Line Interface y la Referencia de comandos de AWS CLI.

Referencia de Windows PowerShell para AWS Device Farm

Para utilizar Windows PowerShell para ejecutar comandos de Device Farm, consulte la referencia de Cmdlet de Device Farm en la AWS Tools for Windows PowerShellreferencia de Cmdlet. Para obtener más información, consulte Configuración de herramientas de AWS para Windows PowerShell en la AWS Tools for Windows PowerShellGuía del usuario.

Automatización de AWS Device Farm

El acceso mediante programación a Device Farm es una forma eficaz de automatizar las tareas comunes que necesita realizar, como programar una ejecución o descargar los artefactos para una ejecución, un conjunto o una prueba. El SDK de AWS y la AWS CLI proporcionan medios para hacerlo.

El SDK de AWS proporciona acceso a todos los servicios de AWS, incluidos Device Farm, Amazon S3 y más. Para obtener más información, consulte

- las herramientas y SDK de AWS
- la referencia de la API de AWS Device Farm

Ejemplo: usar el SDK de AWS para iniciar una ejecución de Device Farm y recopilar artefactos

El siguiente ejemplo proporciona una demostración de principio a fin de cómo puede utilizar el SDK de AWS para trabajar con Device Farm. En el ejemplo se realiza lo siguiente:

- Carga una prueba y paquetes de aplicación en Device Farm
- Inicia una ejecución de prueba y espera su finalización (o fallo)
- Descarga todos los artefactos producidos por los conjuntos de pruebas

Este ejemplo depende del paquete de terceros `requests` para interactuar con 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")
```

Solución de problemas de Device Farm

En esta sección, encontrará mensajes de error y procedimientos para ayudarle a solucionar problemas comunes con Device Farm.

Temas

Solución de problemas de pruebas de aplicaciones Android en AWS Device Farm

En el siguiente tema se muestra una lista de mensajes de error que se producen durante la carga de las pruebas de aplicaciones Android y recomienda soluciones para resolver cada error.

> **Note**
>
> Las siguientes instrucciones se basan en Linux x86_64 y Mac.

ANDROID_APP_UNZIP_FAILED

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not open your application. Please verify that the file is valid and try again.

Asegúrese de que puede descomprimir el paquete de aplicaciones sin errores. En el siguiente ejemplo, el nombre del paquete es app-debug.apk.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip app-debug.apk
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Un paquete de aplicaciones Android válido debería producir una salida similar a esta:

   ```
   .
   |-- AndroidManifest.xml
   |-- classes.dex
   |-- resources.arsc
   |-- assets (directory)
   |-- res (directory)
   `-- META-INF (directory)
   ```

   Para obtener más información, consulte Trabajar con pruebas de Android en AWS Device Farm.

ANDROID_APP_AAPT_DEBUG_BADGING_FAILED

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not extract information about your application. Please verify that the application 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 el proceso de validación de carga, AWS Device Farm extrae la información de la salida de un comando `aapt debug badging <path to your package>`.

Asegúrese de que puede ejecutar correctamente este comando en la aplicación Android. En el siguiente ejemplo, el nombre del paquete es app-debug.apk.

- Copie el paquete de aplicaciones a su directorio de trabajo y, a continuación, ejecute el comando:

  ```
  $ aapt debug badging app-debug.apk
  ```

  Un paquete de aplicaciones Android válido debería producir una salida similar a esta:

  ```
  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 obtener más información, consulte Trabajar con pruebas de Android en AWS Device Farm.

ANDROID_APP_PACKAGE_NAME_VALUE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find the package name value in your application. Please verify that the application 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 el proceso de validación de carga, AWS Device Farm extrae el valor del nombre del paquete de la salida de un comando `aapt debug badging` *`<path to your package>`*.

Asegúrese de que puede ejecutar este comando en la aplicación Android y encontrar el valor del nombre del paquete de forma correcta. En el siguiente ejemplo, el nombre del paquete es app-debug.apk.

- Copie el paquete de aplicaciones a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

  ```
  $ aapt debug badging app-debug.apk | grep "package: name="
  ```

  Un paquete de aplicaciones Android válido debería producir una salida similar a esta:

  ```
  package: name='com.amazon.aws.adf.android.referenceapp' versionCode='1'
   versionName='1.0' platformBuildVersionName='5.1.1-1819727'
  ```

  Para obtener más información, consulte Trabajar con pruebas de Android en AWS Device Farm.

ANDROID_APP_SDK_VERSION_VALUE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find the SDK version value in your application. Please verify that the application is valid by running the command `aapt debug badging` *`<path to your`*

test package>, and try again after finding the SDK version value behind the keyword `sdkVersion`.

durante el proceso de validación de carga, AWS Device Farm extrae el valor de la versión de SDK de la salida de un comando `aapt debug badging` *<path to your package>*.

Asegúrese de que puede ejecutar este comando en la aplicación Android y encontrar el valor del nombre del paquete de forma correcta. En el siguiente ejemplo, el nombre del paquete es app-debug.apk.

- Copie el paquete de aplicaciones a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

  ```
  $ aapt debug badging app-debug.apk | grep "sdkVersion"
  ```

  Un paquete de aplicaciones Android válido debería producir una salida similar a esta:

  ```
  sdkVersion:'9'
  ```

  Para obtener más información, consulte Trabajar con pruebas de Android en AWS Device Farm.

ANDROID_APP_AAPT_DUMP_XMLTREE_FAILED

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find the valid AndroidManifest.xml in your application. 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 el proceso de validación de carga, AWS Device Farm extrae información del árbol de análisis de XML para un archivo XML contenido en el paquete mediante el comando `aapt dump xmltree` *<path to your package>* `AndroidManifest.xml`.

Asegúrese de que puede ejecutar correctamente este comando en la aplicación Android. En el siguiente ejemplo, el nombre del paquete es app-debug.apk.

- Copie el paquete de aplicaciones a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ aapt dump xmltree app-debug.apk. AndroidManifest.xml
```

Un paquete de aplicaciones Android válido debería producir una salida similar a esta:

```
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 obtener más información, consulte Trabajar con pruebas de Android en AWS Device Farm.

ANDROID_APP_DEVICE_ADMIN_PERMISSIONS

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We found that your application requires device admin permissions. Please verify that the permissions are not required by run the command `aapt dump xmltree` *<path to your*

test package> AndroidManifest.xml, and try again after making sure that output does not contain the keyword android.permission.BIND_DEVICE_ADMIN.

durante el proceso de validación de carga, AWS Device Farm extrae información de permisos del árbol de análisis de XML para un archivo XML contenido en el paquete mediante el comando aapt dump xmltree *<path to your package>* AndroidManifest.xml.

Asegúrese de que la aplicación no requiere permiso de administración de dispositivos. En el siguiente ejemplo, el nombre del paquete es app-debug.apk.

- Copie el paquete de aplicaciones a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ aapt dump xmltree app-debug.apk AndroidManifest.xml
```

Debería aparecer una salida como la siguiente:

```
N: android=http://schemas.android.com/apk/res/android
  E: manifest (line=2)
    A: android:versionCode(0x0101021b)=(type 0x10)0x1
    A: android:versionName(0x0101021c)="1.0" (Raw: "1.0")
    A: package="com.amazonaws.devicefarm.android.referenceapp" (Raw:
 "com.amazonaws.devicefarm.android.referenceapp")
    A: platformBuildVersionCode=(type 0x10)0x16 (Raw: "22")
    A: platformBuildVersionName="5.1.1-1819727" (Raw: "5.1.1-1819727")
    E: uses-sdk (line=7)
      A: android:minSdkVersion(0x0101020c)=(type 0x10)0xa
      A: android:targetSdkVersion(0x01010270)=(type 0x10)0x16
    E: uses-permission (line=11)
      A: android:name(0x01010003)="android.permission.INTERNET" (Raw:
 "android.permission.INTERNET")
    E: uses-permission (line=12)
      A: android:name(0x01010003)="android.permission.CAMERA" (Raw:
 "android.permission.CAMERA")
        ……
```

Si la aplicación Android es válida, la salida no debería contener lo siguiente: A: android:name(0x01010003)="android.permission.BIND_DEVICE_ADMIN" (Raw: "android.permission.BIND_DEVICE_ADMIN").

Para obtener más información, consulte Trabajar con pruebas de Android en AWS Device Farm.

Algunas ventanas de mi aplicación de Android muestran una pantalla en blanco o negra

Si está probando una aplicación de Android y observa que algunas ventanas de la aplicación aparecen con una pantalla negra en la grabación de vídeo de la prueba realizada por Device Farm, es posible que la aplicación esté utilizando la característica de Android `FLAG_SECURE`. Este indicador (tal y como se describe en la documentación oficial de Android) se utiliza para impedir que las herramientas de grabación de pantalla graben determinadas ventanas de una aplicación. Como resultado, la característica de grabación de pantalla de Device Farm (tanto para las pruebas de automatización como para las de acceso remoto) puede mostrar una pantalla negra en lugar de la ventana de la aplicación si esta utiliza esta marca.

Los desarrolladores suelen utilizar esta marca para las páginas de sus aplicaciones que contienen información confidencial, como las páginas de inicio de sesión. Si ve una pantalla negra en lugar de la pantalla de la aplicación en determinadas páginas, como la página de inicio de sesión, trabaje con sus desarrolladores para obtener una versión de la aplicación que no utilice esta marca para las pruebas.

Además, tenga en cuenta que Device Farm puede seguir interactuando con las ventanas de aplicaciones que tengan este indicador. Por lo tanto, si la página de inicio de sesión de su aplicación aparece como una pantalla negra, es posible que aún pueda introducir sus credenciales de registro en la aplicación (y así ver las páginas no bloqueadas por la bandera `FLAG_SECURE`).

Solución de problemas de pruebas de Appium Java JUnit en AWS Device Farm

En el siguiente tema se muestra una lista de mensajes de error que se producen durante la carga de las pruebas de Appium Java JUnit y recomienda soluciones para resolver cada error.

> **Note**
>
> Las siguientes instrucciones se basan en Linux x86_64 y Mac.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_PACKAGE_UNZIP_FAILED

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not open your test ZIP file. Please verify that the file is valid and try again.

Asegúrese de que puede descomprimir el paquete de pruebas sin errores. En el siguiente ejemplo, el nombre del paquete es zip-with-dependencies.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Un paquete de Appium Java JUnit válido debería producir una salida similar a esta:

   ```
   .
   |— 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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_DEPENDENCY_DIR_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find the dependency-jars directory inside your test package. Please unzip your test package, verify that the dependency-jars directory is inside the package, and try again.

En el siguiente ejemplo, el nombre del paquete es zip-with-dependencies.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Si el paquete de Appium Java JUnit es válido, encontrará el directorio *dependency-jars* dentro del directorio de trabajo:

   ```
   .
   |— 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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_DI

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find a JAR file in the dependency-jars directory tree. Please unzip your test package and then open the dependency-jars directory, verify that at least one JAR file is in the directory, and try again.

En el siguiente ejemplo, el nombre del paquete es zip-with-dependencies.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Si el paquete de Appium Java JUnit es válido, encontrará como mínimo un archivo *jar* dentro del directorio *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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find a *-tests.jar file in your test package. Please unzip your test package, verify that at least one *-tests.jar file is in the package, and try again.

En el siguiente ejemplo, el nombre del paquete es zip-with-dependencies.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Si el paquete de Appium Java JUnit es válido, encontrará como mínimo un archivo *jar* como *acme-android-appium-1.0-SNAPSHOT-tests.jar* en nuestro ejemplo. El nombre del archivo puede ser diferente, pero debería terminar con *-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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS_JA

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find a class file within the tests JAR file. Please unzip your test package and then unjar the tests JAR file, verify that at least one class file is within the JAR file, and try again.

En el siguiente ejemplo, el nombre del paquete es zip-with-dependencies.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Debería encontrar como mínimo un archivo jar como *acme-android-appium-1.0-SNAPSHOT-tests.jar* en nuestro ejemplo. El nombre del archivo puede ser diferente, pero debería terminar con *-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. Después de extraer correctamente los archivos, debe encontrar al menos una clase en el árbol de directorios de trabajo ejecutando el comando:

```
$ tree .
```

Debería ver una salida similar a esta:

```
.
|— 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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_JUNIT_VERSION_VALUE_UNKNOWN

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find a JUnit version value. Please unzip your test package and open the dependency-jars directory, verify that the JUnit JAR file is inside the directory, and try again.

En el siguiente ejemplo, el nombre del paquete es zip-with-dependencies.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip zip-with-dependencies.zip
```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
tree .
```

La salida debe tener el siguiente aspecto:

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
    |— junit-4.10.jar
    |— com.some-dependency.bar-4.1.jar
    |— com.another-dependency.thing-1.0.jar
    |— joda-time-2.7.jar
    `— log4j-1.2.14.jar
```

Si el paquete de Appium Java JUnit es válido, encontrará el archivo de dependencias de JUnit que es similar al archivo jar *junit-4.10.jar* en nuestro ejemplo. El nombre debería estar compuesto por la palabra clave *junit* y su número de versión, que en este ejemplo es 4.10.

Para obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We found the JUnit version was lower than the minimum version 4.10 we support. Please change the JUnit version and try again.

En el siguiente ejemplo, el nombre del paquete es zip-with-dependencies.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip zip-with-dependencies.zip
```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

Debería encontrar un archivo de dependencias de JUnit como *junit-4.10.jar* en nuestro ejemplo y su número de versión, que en nuestro ejemplo es 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**
>
> Es posible que las pruebas no se ejecuten correctamente si la versión de JUnit especificada en el paquete de pruebas es inferior a la versión mínima admitida que es la 4.10.

Para obtener más información, consulte Trabajar con Appium y AWS Device Farm.

Solución de problemas de las pruebas de una aplicación web de Appium Java JUnit en AWS Device Farm

El siguiente tema muestra una lista de mensajes de error que se producen durante la carga de las pruebas de aplicaciones web de Appium Java JUnit y recomienda soluciones para resolver cada error. Para obtener más información acerca del uso de Appium con Device Farm, consulte the section called “Appium”.

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_UNZIP_FAILED

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not open your test ZIP file. Please verify that the file is valid and try again.

Asegúrese de que puede descomprimir el paquete de pruebas sin errores. En el siguiente ejemplo, el nombre del paquete es zip-with-dependencies.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Un paquete de Appium Java JUnit válido debería producir una salida similar a esta:

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

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

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_DEPENDENCY_DIR_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find the dependency-jars directory inside your test package. Please unzip your test package, verify that the dependency-jars directory is inside the package, and try again.

En el siguiente ejemplo, el nombre del paquete es zip-with-dependencies.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Si el paquete de Appium Java JUnit es válido, encontrará el directorio *dependency-jars* dentro del directorio de trabajo:

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

```
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_JAR_MISSING_IN_DEPENDEN

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find a JAR file in the dependency-jars directory tree. Please unzip your test package and then open the dependency-jars directory, verify that at least one JAR file is in the directory, and try again.

En el siguiente ejemplo, el nombre del paquete es zip-with-dependencies.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Si el paquete de Appium Java JUnit es válido, encontrará como mínimo un archivo *`jar`* dentro del directorio *`dependency-jars`*:

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

```
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find a *-tests.jar file in your test package. Please unzip your test package, verify that at least one *-tests.jar file is in the package, and try again.

En el siguiente ejemplo, el nombre del paquete es zip-with-dependencies.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Si el paquete de Appium Java JUnit es válido, encontrará como mínimo un archivo *jar* como *acme-android-appium-1.0-SNAPSHOT-tests.jar* en nuestro ejemplo. El nombre del archivo puede ser diferente, pero debería terminar con *-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

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find a class file within the tests JAR file. Please unzip your test package and then unjar the tests JAR file, verify that at least one class file is within the JAR file, and try again.

En el siguiente ejemplo, el nombre del paquete es zip-with-dependencies.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Debería encontrar como mínimo un archivo jar como *acme-android-appium-1.0-SNAPSHOT-tests.jar* en nuestro ejemplo. El nombre del archivo puede ser diferente, pero debería terminar con *-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. Después de extraer correctamente los archivos, debe encontrar al menos una clase en el árbol de directorios de trabajo ejecutando el comando:

```
$ tree .
```

Debería ver una salida similar a esta:

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

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_JUNIT_VERSION_VALUE_UNK

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find a JUnit version value. Please unzip your test package and open the dependency-jars directory, verify that the JUnit JAR file is inside the directory, and try again.

En el siguiente ejemplo, el nombre del paquete es zip-with-dependencies.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip zip-with-dependencies.zip
```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
tree .
```

   La salida debe tener el siguiente aspecto:

```
.
|— acme-android-appium-1.0-SNAPSHOT.jar (this is the JAR containing everything
 built from the ./src/main directory)
|— acme-android-appium-1.0-SNAPSHOT-tests.jar (this is the JAR containing
 everything built from the ./src/test directory)
|— zip-with-dependencies.zip (this .zip file contains all of the items)
`— dependency-jars  (this is the directory that contains all of your dependencies,
 built as JAR files)
      |— junit-4.10.jar
      |— com.some-dependency.bar-4.1.jar
      |— com.another-dependency.thing-1.0.jar
      |— joda-time-2.7.jar
      `— log4j-1.2.14.jar
```

   Si el paquete de Appium Java JUnit es válido, encontrará el archivo de dependencias de JUnit que es similar al archivo jar *junit-4.10.jar* en nuestro ejemplo. El nombre debería estar compuesto por la palabra clave *junit* y su número de versión, que en este ejemplo es 4.10.

APPIUM_WEB_JAVA_JUNIT_TEST_PACKAGE_INVALID_JUNIT_VERSION

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We found the JUnit version was lower than the minimum version 4.10 we support. Please change the JUnit version and try again.

En el siguiente ejemplo, el nombre del paquete es zip-with-dependencies.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip zip-with-dependencies.zip
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Debería encontrar un archivo de dependencias de JUnit como *junit-4.10.jar* en nuestro ejemplo y su número de versión, que en nuestro ejemplo es 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**
>
> Es posible que las pruebas no se ejecuten correctamente si la versión de JUnit especificada en el paquete de pruebas es inferior a la versión mínima admitida que es la 4.10.

Para obtener más información, consulte Trabajar con Appium y AWS Device Farm.

Solución de problemas de pruebas de Appium Java TestNG en AWS Device Farm

En el siguiente tema se muestra una lista de mensajes de error que se producen durante la carga de las pruebas de Appium Java TestNG y recomienda soluciones para resolver cada error.

> **Note**
>
> Las siguientes instrucciones se basan en Linux x86_64 y Mac.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not open your test ZIP file. Please verify that the file is valid and try again.

Asegúrese de que puede descomprimir el paquete de pruebas sin errores. En el siguiente ejemplo, el nombre del paquete es zip-with-dependencies.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip zip-with-dependencies.zip
```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

Un paquete de Appium Java JUnit válido debería producir una salida similar a esta:

```
.
|— 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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find the `dependency-jars` directory inside your test package. Please unzip your test package, verify that the `dependency-jars` directory is inside the package, and try again.

En el siguiente ejemplo, el nombre del paquete es zip-with-dependencies.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip zip-with-dependencies.zip
```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

Si el paquete de Appium Java JUnit es válido, encontrará el directorio *dependency-jars* dentro del directorio de trabajo.

```
.
|— 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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDENCY_

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find a JAR file in the dependency-jars directory tree. Please unzip your test package and then open the dependency-jars directory, verify that at least one JAR file is in the directory, and try again.

En el siguiente ejemplo, el nombre del paquete es zip-with-dependencies.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip zip-with-dependencies.zip
```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

Si el paquete de Appium Java JUnit es válido, encontrará como mínimo un archivo *jar* dentro del directorio *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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find a *-tests.jar file in your test package. Please unzip your test package, verify that at least one *-tests.jar file is in the package, and try again.

En el siguiente ejemplo, el nombre del paquete es zip-with-dependencies.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip zip-with-dependencies.zip
```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

Si el paquete de Appium Java JUnit es válido, encontrará como mínimo un archivo *jar* como *acme-android-appium-1.0-SNAPSHOT-tests.jar* en nuestro ejemplo. El nombre del archivo puede ser diferente, pero debería terminar con *-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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_TESTS

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find a class file within the tests JAR file. Please unzip your test package and then unjar the tests JAR file, verify that at least one class file is within the JAR file, and try again.

En el siguiente ejemplo, el nombre del paquete es zip-with-dependencies.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip zip-with-dependencies.zip
```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

Debería encontrar como mínimo un archivo jar como *acme-android-appium-1.0-SNAPSHOT-tests.jar* en nuestro ejemplo. El nombre del archivo puede ser diferente, pero debería terminar con *-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 extraer archivos del archivo jar, puede ejecutar el siguiente comando:

```
$ jar xf acme-android-appium-1.0-SNAPSHOT-tests.jar
```

4. Después de extraer correctamente los archivos, ejecute el siguiente comando:

```
$ tree .
```

Debería encontrar una clase como mínimo en el árbol del directorio de trabajo:

```
.
|— 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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

Solución de problemas de las pruebas de una aplicación web de Appium Java TestNG en AWS Device Farm

El siguiente tema muestra una lista de mensajes de error que se producen durante la carga de las pruebas de aplicaciones web de Appium Java TestNG y recomienda soluciones para resolver cada error.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_UNZIP_FAILED

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not open your test ZIP file. Please verify that the file is valid and try again.

Asegúrese de que puede descomprimir el paquete de pruebas sin errores. En el siguiente ejemplo, el nombre del paquete es zip-with-dependencies.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip zip-with-dependencies.zip
```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

Un paquete de Appium Java JUnit válido debería producir una salida similar a esta:

```
.
|— 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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_DEPENDENCY_DIR_MISSIN

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find the dependency-jars directory inside your test package. Please unzip your test package, verify that the dependency-jars directory is inside the package, and try again.

En el siguiente ejemplo, el nombre del paquete es zip-with-dependencies.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip zip-with-dependencies.zip
```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

Si el paquete de Appium Java JUnit es válido, encontrará el directorio *dependency-jars* dentro del directorio de trabajo.

```
.
|— 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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPENDE

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find a JAR file in the dependency-jars directory tree. Please unzip your test package and then open the dependency-jars directory, verify that at least one JAR file is in the directory, and try again.

En el siguiente ejemplo, el nombre del paquete es zip-with-dependencies.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip zip-with-dependencies.zip
```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

Si el paquete de Appium Java JUnit es válido, encontrará como mínimo un archivo *jar* dentro del directorio *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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_TESTS_JAR_FILE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find a *-tests.jar file in your test package. Please unzip your test package, verify that at least one *-tests.jar file is in the package, and try again.

En el siguiente ejemplo, el nombre del paquete es zip-with-dependencies.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip zip-with-dependencies.zip
```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

Si el paquete de Appium Java JUnit es válido, encontrará como mínimo un archivo *jar* como *acme-android-appium-1.0-SNAPSHOT-tests.jar* en nuestro ejemplo. El nombre del archivo puede ser diferente, pero debería terminar con *-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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_CLASS_FILE_MISSING_IN_T

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find a class file within the tests JAR file. Please unzip your test package and then unjar the tests JAR file, verify that at least one class file is within the JAR file, and try again.

En el siguiente ejemplo, el nombre del paquete es zip-with-dependencies.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip zip-with-dependencies.zip
```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

Debería encontrar como mínimo un archivo jar como *acme-android-appium-1.0-SNAPSHOT-tests.jar* en nuestro ejemplo. El nombre del archivo puede ser diferente, pero debería terminar con *-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 extraer archivos del archivo jar, puede ejecutar el siguiente comando:

```
$ jar xf acme-android-appium-1.0-SNAPSHOT-tests.jar
```

4. Después de extraer correctamente los archivos, ejecute el siguiente comando:

```
$ tree .
```

Debería encontrar una clase como mínimo en el árbol del directorio de trabajo:

```
.
|— 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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

Solución de problemas de pruebas de Python de Appium en AWS Device Farm

En el siguiente tema se muestra una lista de mensajes de error que se producen durante la carga de las pruebas de Appium Python y recomienda soluciones para resolver cada error.

APPIUM_PYTHON_TEST_PACKAGE_UNZIP_FAILED

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not open your Appium test ZIP file. Please verify that the file is valid and try again.

Asegúrese de que puede descomprimir el paquete de pruebas sin errores. En el siguiente ejemplo, el nombre del paquete es test_bundle.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip test_bundle.zip
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

Un paquete de Appium Python válido debería producir una salida similar a esta:

```
.
|-- 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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find a dependency wheel file in the wheelhouse directory tree. Please unzip your test package and then open the wheelhouse directory, verify that at least one wheel file is in the directory, and try again.

Asegúrese de que puede descomprimir el paquete de pruebas sin errores. En el siguiente ejemplo, el nombre del paquete es test_bundle.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip test_bundle.zip
```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

Si el paquete de Appium Python es válido, encontrará como mínimo un archivo *.whl* dependiente como los archivos resaltados dentro del directorio *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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_INVALID_PLATFORM

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We found at least one wheel file specified a platform that we do not support. Please unzip your test package and then open the wheelhouse directory, verify that names of wheel files end with -any.whl or -linux_x86_64.whl, and try again.

Asegúrese de que puede descomprimir el paquete de pruebas sin errores. En el siguiente ejemplo, el nombre del paquete es test_bundle.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip test_bundle.zip
```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

Si el paquete de Appium Python es válido, encontrará como mínimo un archivo *.whl* dependiente como los archivos resaltados dentro del directorio *wheelhouse*. El nombre del archivo puede ser diferente, pero debería terminar con *-any.whl* o *-linux_x86_64.whl*, que especifica la plataforma. No se admite ninguna otra plataforma, 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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find the tests directory inside your test package. Please unzip your test package, verify that the tests directory is inside the package, and try again.

Asegúrese de que puede descomprimir el paquete de pruebas sin errores. En el siguiente ejemplo, el nombre del paquete es test_bundle.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip test_bundle.zip
```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

Si el paquete de Appium Python es válido, encontrará el directorio *tests* dentro del directorio de trabajo.

```
.
|-- 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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find a valid test file in the tests directory tree. Please unzip your test package and then open the tests directory, verify that at least one file's name starts or ends with the keyword "test", and try again.

Asegúrese de que puede descomprimir el paquete de pruebas sin errores. En el siguiente ejemplo, el nombre del paquete es test_bundle.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip test_bundle.zip
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Si el paquete de Appium Python es válido, encontrará el directorio *tests* dentro del directorio de trabajo. El nombre del archivo puede ser diferente, pero debería comenzar con *test_* o terminar con *_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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find the requirements.txt file inside your test package. Please unzip your test package, verify that the requirements.txt file is inside the package, and try again.

Asegúrese de que puede descomprimir el paquete de pruebas sin errores. En el siguiente ejemplo, el nombre del paquete es test_bundle.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip test_bundle.zip
```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

Si el paquete de Appium Python es válido, encontrará el archivo *requirements.txt* dentro del directorio de trabajo.

```
.
|-- 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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We found the pytest version was lower than the minimum version 2.8.0 we support. Please change the pytest version inside the requirements.txt file, and try again.

Asegúrese de que puede descomprimir el paquete de pruebas sin errores. En el siguiente ejemplo, el nombre del paquete es test_bundle.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip test_bundle.zip
```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

   Debería encontrar el archivo *requirements.txt* dentro del directorio de trabajo.

```
.
|-- 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 obtener la versión de pytest, ejecute el siguiente comando:

```
$ grep "pytest" requirements.txt
```

   Debería aparecer una salida como la siguiente:

```
pytest==2.9.0
```

   Muestra la versión pytest, que en este ejemplo es 2.9.0. Si el paquete de Appium Python es válido, la versión de pytest debe ser mayor o igual que 2.8.0.

   Para obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS_FAIL

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We failed to install the dependency wheels. Please unzip your test package and then open the requirements.txt file and the wheelhouse directory, verify that the dependency wheels specified in the requirements.txt file exactly match the dependency wheels inside the wheelhouse directory, and try again.

Le recomendamos encarecidamente que configure virtualenv de Python para el empaquetamiento de pruebas. A continuación se muestra un ejemplo de flujo para crear un entorno virtual con Python virtualenv y, a continuación, activarlo:

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

Asegúrese de que puede descomprimir el paquete de pruebas sin errores. En el siguiente ejemplo, el nombre del paquete es test_bundle.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip test_bundle.zip
   ```

2. Para probar la instalación de archivos wheel, puede ejecutar el siguiente comando:

   ```
   $ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
   requirements.txt
   ```

   Un paquete de Appium Python válido debería producir una salida similar a esta:

   ```
   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 desactivar el entorno virtual, puede ejecutar el siguiente comando:

```
$ deactivate
```

Para obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We failed to collect tests in the tests directory. Please unzip your test package, very that the test package is valid by running the command `py.test --collect-only <path to your tests directory>`, and try again after the command does not print any error.

Le recomendamos encarecidamente que configure virtualenv de Python para el empaquetamiento de pruebas. A continuación se muestra un ejemplo de flujo para crear un entorno virtual con Python virtualenv y, a continuación, activarlo:

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

Asegúrese de que puede descomprimir el paquete de pruebas sin errores. En el siguiente ejemplo, el nombre del paquete es test_bundle.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip test_bundle.zip
```

2. Para instalar los archivos wheel, puede ejecutar el siguiente comando:

```
$ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
requirements.txt
```

3. Para recopilar pruebas puede ejecutar el siguiente comando:

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

Un paquete de Appium Python válido debería producir una salida similar a esta:

```
==================== 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 desactivar el entorno virtual, puede ejecutar el siguiente comando:

```
$ deactivate
```

Para obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEELS_INSUFFICIENT

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> No hemos podido encontrar suficientes dependencias entre ruedas en el directorio wheelhouse. Descomprima el paquete de prueba y, a continuación, abra el directorio

wheelhouse. Compruebe que tiene todas las dependencias entre ruedas especificadas en el archivo requirements.txt.

Asegúrese de que puede descomprimir el paquete de pruebas sin errores. En el siguiente ejemplo, el nombre del paquete es test_bundle.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip test_bundle.zip
```

2. Compruebe la longitud del archivo *requirements.txt* y el número de archivos dependientes *de.whl* en el directorio wheelhouse:

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

   Si el número de archivos dependientes de *.whl* es inferior al número de filas no vacías del archivo *requirements.txt*, asegúrese de lo siguiente:

   - *Hay un archivo dependiente de .whl correspondiente a cada fila del archivo requirements.txt.*
   - No hay otras líneas en el archivo *requirements.txt* que contengan información distinta de los nombres de los paquetes de dependencias.
   - Los nombres de las dependencias no están duplicados en varias líneas del archivo *requirements.txt*, por lo que dos líneas del archivo pueden corresponder a un archivo dependiente de *.whl*.

   AWS Device Farm no admite líneas en el archivo *requirements.txt* que no se correspondan directamente con los paquetes de dependencias, como las líneas que especifican las opciones globales del `pip install` comando. Consulte el formato de archivo de requisitos para ver una lista de opciones globales.

   Para obtener más información, consulte Trabajar con Appium y AWS Device Farm.

Solución de problemas de pruebas de Python de Appium en AWS Device Farm

El siguiente tema muestra una lista de mensajes de error que se producen durante la carga de las pruebas de aplicaciones web de Appium Python y recomienda soluciones para resolver cada error.

APPIUM_WEB_PYTHON_TEST_PACKAGE_UNZIP_FAILED

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not open your Appium test ZIP file. Please verify that the file is valid and try again.

Asegúrese de que puede descomprimir el paquete de pruebas sin errores. En el siguiente ejemplo, el nombre del paquete es test_bundle.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip test_bundle.zip
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Un paquete de Appium Python válido debería producir una salida similar a esta:

   ```
   .
   |-- 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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEEL_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find a dependency wheel file in the wheelhouse directory tree. Please unzip your test package and then open the wheelhouse directory, verify that at least one wheel file is in the directory, and try again.

Asegúrese de que puede descomprimir el paquete de pruebas sin errores. En el siguiente ejemplo, el nombre del paquete es test_bundle.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip test_bundle.zip
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Si el paquete de Appium Python es válido, encontrará como mínimo un archivo *.whl* dependiente como los archivos resaltados dentro del directorio *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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PLATFORM

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We found at least one wheel file specified a platform that we do not support. Please unzip your test package and then open the wheelhouse directory, verify that names of wheel files end with -any.whl or -linux_x86_64.whl, and try again.

Asegúrese de que puede descomprimir el paquete de pruebas sin errores. En el siguiente ejemplo, el nombre del paquete es test_bundle.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip test_bundle.zip
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Si el paquete de Appium Python es válido, encontrará como mínimo un archivo *.whl* dependiente como los archivos resaltados dentro del directorio *wheelhouse*. El nombre del archivo puede ser diferente, pero debería terminar con *-any.whl* o *-linux_x86_64.whl*, que especifica la plataforma. No se admite ninguna otra plataforma, 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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_TEST_DIR_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find the tests directory inside your test package. Please unzip your test package, verify that the tests directory is inside the package, and try again.

Asegúrese de que puede descomprimir el paquete de pruebas sin errores. En el siguiente ejemplo, el nombre del paquete es test_bundle.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip test_bundle.zip
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Si el paquete de Appium Python es válido, encontrará el directorio *tests* dentro del directorio de trabajo.

   ```
   .
   |-- 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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_TEST_FILE_NAME

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find a valid test file in the tests directory tree. Please unzip your test package and then open the tests directory, verify that at least one file's name starts or ends with the keyword "test", and try again.

Asegúrese de que puede descomprimir el paquete de pruebas sin errores. En el siguiente ejemplo, el nombre del paquete es test_bundle.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip test_bundle.zip
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Si el paquete de Appium Python es válido, encontrará el directorio *tests* dentro del directorio de trabajo. El nombre del archivo puede ser diferente, pero debería comenzar con *test_* o terminar con *_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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_REQUIREMENTS_TXT_FILE_MISS

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find the requirements.txt file inside your test package. Please unzip your test package, verify that the requirements.txt file is inside the package, and try again.

Asegúrese de que puede descomprimir el paquete de pruebas sin errores. En el siguiente ejemplo, el nombre del paquete es test_bundle.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip test_bundle.zip
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

Si el paquete de Appium Python es válido, encontrará el archivo *requirements.txt* dentro del directorio de trabajo.

```
.
|-- 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 obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_INVALID_PYTEST_VERSION

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We found the pytest version was lower than the minimum version 2.8.0 we support. Please change the pytest version inside the requirements.txt file, and try again.

Asegúrese de que puede descomprimir el paquete de pruebas sin errores. En el siguiente ejemplo, el nombre del paquete es test_bundle.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip test_bundle.zip
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

Debería encontrar el archivo *requirement.txt* dentro del directorio de trabajo.

```
.
|-- 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 obtener la versión de pytest, ejecute el siguiente comando:

```
$ grep "pytest" requirements.txt
```

Debería aparecer una salida como la siguiente:

```
pytest==2.9.0
```

Muestra la versión pytest, que en este ejemplo es 2.9.0. Si el paquete de Appium Python es válido, la versión de pytest debe ser mayor o igual que 2.8.0.

Para obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_INSTALL_DEPENDENCY_WHEELS

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We failed to install the dependency wheels. Please unzip your test package and then open the requirements.txt file and the wheelhouse directory, verify that the dependency

wheels specified in the requirements.txt file exactly match the dependency wheels inside the wheelhouse directory, and try again.

Le recomendamos encarecidamente que configure virtualenv de Python para el empaquetamiento de pruebas. A continuación se muestra un ejemplo de flujo para crear un entorno virtual con Python virtualenv y, a continuación, activarlo:

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

Asegúrese de que puede descomprimir el paquete de pruebas sin errores. En el siguiente ejemplo, el nombre del paquete es test_bundle.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip test_bundle.zip
   ```

2. Para probar la instalación de archivos wheel, puede ejecutar el siguiente comando:

   ```
   $ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
   requirements.txt
   ```

   Un paquete de Appium Python válido debería producir una salida similar a esta:

   ```
   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 desactivar el entorno virtual, puede ejecutar el siguiente comando:

```
$ deactivate
```

Para obtener más información, consulte Trabajar con Appium y AWS Device Farm.

APPIUM_WEB_PYTHON_TEST_PACKAGE_PYTEST_COLLECT_FAILED

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We failed to collect tests in the tests directory. Please unzip your test package, very that the test package is valid by running the command "py.test --collect-only <path to your tests directory>", and try again after the command does not print any error.

Le recomendamos encarecidamente que configure virtualenv de Python para el empaquetamiento de pruebas. A continuación se muestra un ejemplo de flujo para crear un entorno virtual con Python virtualenv y, a continuación, activarlo:

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

Asegúrese de que puede descomprimir el paquete de pruebas sin errores. En el siguiente ejemplo, el nombre del paquete es test_bundle.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip test_bundle.zip
   ```

2. Para instalar los archivos wheel, puede ejecutar el siguiente comando:

   ```
   $ pip install --use-wheel --no-index --find-links=./wheelhouse --requirement=./
   requirements.txt
   ```

3. Para recopilar pruebas puede ejecutar el siguiente comando:

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

Un paquete de Appium Python válido debería producir una salida similar a esta:

```
=================== 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 desactivar el entorno virtual, puede ejecutar el siguiente comando:

```
$ deactivate
```

Para obtener más información, consulte Trabajar con Appium y AWS Device Farm.

Solución de problemas de pruebas de instrumentación en AWS Device Farm

En el siguiente tema se muestra una lista de mensajes de error que se producen durante la carga de las pruebas de instrumentación y recomienda soluciones para resolver cada error.

INSTRUMENTATION_TEST_PACKAGE_UNZIP_FAILED

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not open your test APK file. Please verify that the file is valid and try again.

Asegúrese de que puede descomprimir el paquete de pruebas sin errores. En el siguiente ejemplo, el nombre del paquete es app-debug-androidTest-unaligned.apk.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip app-debug-androidTest-unaligned.apk
```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

Un paquete de pruebas de instrumentación válido producirá una salida similar a esta:

```
.
|-- AndroidManifest.xml
|--  classes.dex
|-- resources.arsc
|-- LICENSE-junit.txt
|-- junit (directory)
`-- META-INF (directory)
```

Para obtener más información, consulte Trabajar con instrumentación para Android y AWS Device Farm.

INSTRUMENTATION_TEST_PACKAGE_AAPT_DEBUG_BADGING_FAILED

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> 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 el proceso de validación de carga, Device Farm extrae la información de la salida del comando `aapt debug badging <path to your package>`.

Asegúrese de que puede ejecutar correctamente este comando en el paquete de pruebas de instrumentación.

En el siguiente ejemplo, el nombre del paquete es app-debug-androidTest-unaligned.apk.

- Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ aapt debug badging app-debug-androidTest-unaligned.apk
```

Un paquete de pruebas de instrumentación válido producirá una salida similar a esta:

```
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 obtener más información, consulte Trabajar con instrumentación para Android y AWS Device Farm.

INSTRUMENTATION_TEST_PACKAGE_INSTRUMENTATION_RUNNER_VALU

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> 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 el proceso de validación de carga, Device Farm extrae el valor de la aplicación de ejecución de instrumentación del árbol de análisis de XML para un archivo XML contenido en el paquete. Puede utilizar el siguiente comando: `aapt dump xmltree <path to your package> AndroidManifest.xml`.

Asegúrese de que puede ejecutar este comando en el paquete de pruebas de instrumentación y encontrar el valor de la instrumentación de forma correcta.

En el siguiente ejemplo, el nombre del paquete es app-debug-androidTest-unaligned.apk.

- Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ aapt dump xmltree app-debug-androidTest-unaligned.apk AndroidManifest.xml | grep
 -A5 "instrumentation"
```

Un paquete de pruebas de instrumentación válido producirá una salida similar a esta:

```
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 obtener más información, consulte Trabajar con instrumentación para Android y AWS Device Farm.

INSTRUMENTATION_TEST_PACKAGE_AAPT_DUMP_XMLTREE_FAILED

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> 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 el proceso de validación de carga, Device Farm extrae información del árbol de análisis de XML para un archivo XML contenido en el paquete mediante el siguiente comando: `aapt dump xmltree <path to your package> AndroidManifest.xml`.

Asegúrese de que puede ejecutar correctamente este comando en el paquete de pruebas de instrumentación.

En el siguiente ejemplo, el nombre del paquete es app-debug-androidTest-unaligned.apk.

- Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

  ```
  $ aapt dump xmltree app-debug-androidTest-unaligned.apk AndroidManifest.xml
  ```

  Un paquete de pruebas de instrumentación válido producirá una salida similar a esta:

  ```
  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 obtener más información, consulte Trabajar con instrumentación para Android y AWS Device Farm.

INSTRUMENTATION_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISS

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> 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 el proceso de validación de carga, Device Farm extrae el valor del nombre del paquete de la salida del siguiente comando: `aapt debug badging <path to your package>`.

Asegúrese de que puede ejecutar este comando en el paquete de pruebas de instrumentación y encontrar el valor del nombre del paquete de forma correcta.

En el siguiente ejemplo, el nombre del paquete es app-debug-androidTest-unaligned.apk.

- Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

  ```
  $ aapt debug badging app-debug-androidTest-unaligned.apk | grep "package: name="
  ```

  Un paquete de pruebas de instrumentación válido producirá una salida similar a esta:

```
package: name='com.amazon.aws.adf.android.referenceapp.test' versionCode=''
 versionName='' platformBuildVersionName='5.1.1-1819727'
```

Para obtener más información, consulte Trabajar con instrumentación para Android y AWS Device Farm.

Solución de problemas de pruebas de aplicaciones iOS en AWS Device Farm

En el siguiente tema se muestra una lista de mensajes de error que se producen durante la carga de las pruebas de aplicaciones iOS y recomienda soluciones para resolver cada error.

> **Note**
>
> Las siguientes instrucciones se basan en Linux x86_64 y Mac.

IOS_APP_UNZIP_FAILED

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not open your application. Please verify that the file is valid and try again.

Asegúrese de que puede descomprimir el paquete de aplicaciones sin errores. En el siguiente ejemplo, el nombre del paquete es AWSDeviceFarmiOSReferenceApp.ipa.

1. Copie el paquete de aplicaciones a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

Un paquete de aplicaciones iOS válido debería producir una salida similar a esta:

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

Para obtener más información, consulte Uso de pruebas de iOS en AWS Device Farm.

IOS_APP_PAYLOAD_DIR_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find the Payload directory inside your application. Please unzip your application, verify that the Payload directory is inside the package, and try again.

En el siguiente ejemplo, el nombre del paquete es AWSDeviceFarmiOSReferenceApp.ipa.

1. Copie el paquete de aplicaciones a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Si el paquete de aplicaciones iOS es válido, encontrará el directorio *Payload* dentro del directorio de trabajo.

   ```
   .
   ```

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

Para obtener más información, consulte Uso de pruebas de iOS en AWS Device Farm.

IOS_APP_APP_DIR_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find the .app directory inside the Payload directory. Please unzip your application and then open the Payload directory, verify that the .app directory is inside the directory, and try again.

En el siguiente ejemplo, el nombre del paquete es AWSDeviceFarmiOSReferenceApp.ipa.

1. Copie el paquete de aplicaciones a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Si el paquete de aplicaciones iOS es válido, encontrará un directorio *.app*, como *AWSDeviceFarmiOSReferenceApp.app* en nuestro ejemplo, dentro del directorio *Payload*.

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

Para obtener más información, consulte Uso de pruebas de iOS en AWS Device Farm.

IOS_APP_PLIST_FILE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find the Info.plist file inside the .app directory. Please unzip your application and then open the .app directory, verify that the Info.plist file is inside the directory, and try again.

En el siguiente ejemplo, el nombre del paquete es AWSDeviceFarmiOSReferenceApp.ipa.

1. Copie el paquete de aplicaciones a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Si el paquete de aplicaciones iOS es válido, encontrará el archivo *Info.plist* dentro del directorio *.app*, como *AWSDeviceFarmiOSReferenceApp.app* en nuestro ejemplo.

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

   Para obtener más información, consulte Uso de pruebas de iOS en AWS Device Farm.

IOS_APP_CPU_ARCHITECTURE_VALUE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find the CPU architecture value in the Info.plist file. Please unzip your application and then open Info.plist file inside the .app directory, verify that the key "UIRequiredDeviceCapabilities" is specified, and try again.

En el siguiente ejemplo, el nombre del paquete es AWSDeviceFarmiOSReferenceApp.ipa.

1. Copie el paquete de aplicaciones a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Debe encontrar el archivo *Info.plist* dentro de un directorio *.app*, como *AWSDeviceFarmiOSReferenceApp.app* en nuestro ejemplo:

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

3. Para encontrar el valor de la arquitectura de la CPU, puede abrir Info.plist mediante Xcode o Python.

   Para Python, puede instalar el módulo biplist ejecutando el siguiente comando:

   ```
   $ pip install biplist
   ```

4. A continuación, abra Python y ejecute el siguiente comando:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
   Info.plist')
   ```

```
print info_plist['UIRequiredDeviceCapabilities']
```

Un paquete de aplicaciones iOS válido debería producir una salida similar a esta:

```
['armv7']
```

Para obtener más información, consulte Uso de pruebas de iOS en AWS Device Farm.

IOS_APP_PLATFORM_VALUE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find the platform value in the Info.plist file. Please unzip your application and then open Info.plist file inside the .app directory, verify that the key "CFBundleSupportedPlatforms" is specified, and try again.

En el siguiente ejemplo, el nombre del paquete es AWSDeviceFarmiOSReferenceApp.ipa.

1. Copie el paquete de aplicaciones a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Debe encontrar el archivo *Info.plist* dentro de un directorio *.app*, como *AWSDeviceFarmiOSReferenceApp.app* en nuestro ejemplo:

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

```
                        `-- (any other files)
```

3. Para encontrar el valor de la plataforma, puede abrir Info.plist mediante Xcode o Python.

   Para Python, puede instalar el módulo biplist ejecutando el siguiente comando:

   ```
   $ pip install biplist
   ```

4. A continuación, abra Python y ejecute el siguiente comando:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
   Info.plist')
   print info_plist['CFBundleSupportedPlatforms']
   ```

   Un paquete de aplicaciones iOS válido debería producir una salida similar a esta:

   ```
   ['iPhoneOS']
   ```

   Para obtener más información, consulte Uso de pruebas de iOS en AWS Device Farm.

IOS_APP_WRONG_PLATFORM_DEVICE_VALUE

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We found the platform device value was wrong in the Info.plist file. Please unzip your application and then open Info.plist file inside the .app directory, verify that the value of the key "CFBundleSupportedPlatforms" does not contain the keyword "simulator", and try again.

En el siguiente ejemplo, el nombre del paquete es AWSDeviceFarmiOSReferenceApp.ipa.

1. Copie el paquete de aplicaciones a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

Debe encontrar el archivo *Info.plist* dentro de un directorio *.app*, como *AWSDeviceFarmiOSReferenceApp.app* en nuestro ejemplo:

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

3. Para encontrar el valor de la plataforma, puede abrir Info.plist mediante Xcode o Python.

Para Python, puede instalar el módulo biplist ejecutando el siguiente comando:

```
$ pip install biplist
```

4. A continuación, abra Python y ejecute el siguiente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
Info.plist')
print info_plist['CFBundleSupportedPlatforms']
```

Un paquete de aplicaciones iOS válido debería producir una salida similar a esta:

```
['iPhoneOS']
```

Si la aplicación iOS es válida, el valor no debería contener la palabra clave `simulator`.

Para obtener más información, consulte Uso de pruebas de iOS en AWS Device Farm.

IOS_APP_FORM_FACTOR_VALUE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find the form factor value in the Info.plist file. Please unzip your application and then open Info.plist file inside the .app directory, verify that the key "UIDeviceFamily" is specified, and try again.

En el siguiente ejemplo, el nombre del paquete es AWSDeviceFarmiOSReferenceApp.ipa.

1. Copie el paquete de aplicaciones a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Debe encontrar el archivo *Info.plist* dentro de un directorio *.app*, como *AWSDeviceFarmiOSReferenceApp.app* en nuestro ejemplo:

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

3. Para encontrar el valor del factor de forma, puede abrir Info.plist mediante Xcode o Python.

   Para Python, puede instalar el módulo biplist ejecutando el siguiente comando:

   ```
   $ pip install biplist
   ```

4. A continuación, abra Python y ejecute el siguiente comando:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
   Info.plist')
   print info_plist['UIDeviceFamily']
   ```

Un paquete de aplicaciones iOS válido debería producir una salida similar a esta:

```
[1, 2]
```

Para obtener más información, consulte Uso de pruebas de iOS en AWS Device Farm.

IOS_APP_PACKAGE_NAME_VALUE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find the package ñame value in the Info.plist file. Please unzip your application and then open Info.plist file inside the .app directory, verify that the key "CFBundleIdentifier" is specified, and try again.

En el siguiente ejemplo, el nombre del paquete es AWSDeviceFarmiOSReferenceApp.ipa.

1. Copie el paquete de aplicaciones a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Debe encontrar el archivo *Info.plist* dentro de un directorio *.app*, como *AWSDeviceFarmiOSReferenceApp.app* en nuestro ejemplo:

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

3. Para encontrar el valor del nombre del paquete, puede abrir Info.plist mediante Xcode o Python.

   Para Python, puede instalar el módulo biplist ejecutando el siguiente comando:

   ```
   $ pip install biplist
   ```

4. A continuación, abra Python y ejecute el siguiente comando:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
   Info.plist')
   print info_plist['CFBundleIdentifier']
   ```

   Un paquete de aplicaciones iOS válido debería producir una salida similar a esta:

   ```
   Amazon.AWSDeviceFarmiOSReferenceApp
   ```

   Para obtener más información, consulte Uso de pruebas de iOS en AWS Device Farm.

IOS_APP_EXECUTABLE_VALUE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find the executable value in the Info.plist file. Please unzip your application and then open Info.plist file inside the .app directory, verify that the key "CFBundleExecutable" is specified, and try again.

En el siguiente ejemplo, el nombre del paquete es AWSDeviceFarmiOSReferenceApp.ipa.

1. Copie el paquete de aplicaciones a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip AWSDeviceFarmiOSReferenceApp.ipa
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

Debe encontrar el archivo *Info.plist* dentro de un directorio *.app*, como *AWSDeviceFarmiOSReferenceApp.app* en nuestro ejemplo:

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

3. Para encontrar el valor del ejecutable puede abrir Info.plist mediante Xcode o Python.

   Para Python, puede instalar el módulo biplist ejecutando el siguiente comando:

   ```
   $ pip install biplist
   ```

4. A continuación, abra Python y ejecute el siguiente comando:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/AWSDeviceFarmiOSReferenceApp-cal.app/
   Info.plist')
   print info_plist['CFBundleExecutable']
   ```

   Un paquete de aplicaciones iOS válido debería producir una salida similar a esta:

   ```
   AWSDeviceFarmiOSReferenceApp
   ```

   Para obtener más información, consulte Uso de pruebas de iOS en AWS Device Farm.

Solución de problemas de pruebas de XCTest en AWS Device Farm

En el siguiente tema se muestra una lista de mensajes de error que se producen durante la carga de las pruebas de XCTest y recomienda soluciones para resolver cada error.

> **Note**
>
> Las instrucciones que aparecen a continuación suponen que utiliza MacOS.

XCTEST_TEST_PACKAGE_UNZIP_FAILED

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not open your test ZIP file. Please verify that the file is valid and try again.

Asegúrese de que puede descomprimir el paquete de aplicaciones sin errores. En el siguiente ejemplo, el nombre del paquete es swiftExampleTests.xctest-1.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip swiftExampleTests.xctest-1.zip
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Un paquete de XCTest válido debería producir una salida similar a esta:

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

   Para obtener más información, consulte Uso de XCTest para iOS y AWS Device Farm.

XCTEST_TEST_PACKAGE_XCTEST_DIR_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find the .xctest directory inside your test package. Please unzip your test package, verify that the .xctest directory is inside the package, and try again.

En el siguiente ejemplo, el nombre del paquete es swiftExampleTests.xctest-1.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip swiftExampleTests.xctest-1.zip
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Si el paquete de XCTest es válido, encontrará un directorio con un nombre similar a *swiftExampleTests.xctest* dentro del directorio de trabajo. El nombre debe terminar con *.xctest*.

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

   Para obtener más información, consulte Uso de XCTest para iOS y AWS Device Farm.

XCTEST_TEST_PACKAGE_PLIST_FILE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> 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.

En el siguiente ejemplo, el nombre del paquete es swiftExampleTests.xctest-1.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

Si el paquete de XCTest es válido, encontrará el archivo *Info.plist* dentro del directorio de trabajo *.xctest*. En el ejemplo siguiente, el directorio se denomina *swiftExampleTests.xctest*.

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

Para obtener más información, consulte Uso de XCTest para iOS y AWS Device Farm.

XCTEST_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find the package ñame value in the Info.plist file. Please unzip your test package and then open Info.plist file, verify that the key "CFBundleIdentifier" is specified, and try again.

En el siguiente ejemplo, el nombre del paquete es swiftExampleTests.xctest-1.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip swiftExampleTests.xctest-1.zip
```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

Debe encontrar el archivo *Info.plist* dentro de un directorio *.xctest*, como *swiftExampleTests.xctest* en nuestro ejemplo:

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

3. Para encontrar el valor del nombre del paquete, puede abrir Info.plist mediante Xcode o Python.

Para Python, puede instalar el módulo biplist ejecutando el siguiente comando:

```
$ pip install biplist
```

4. A continuación, abra Python y ejecute el siguiente comando:

```
import biplist
info_plist = biplist.readPlist('swiftExampleTests.xctest/Info.plist')
print info_plist['CFBundleIdentifier']
```

Un paquete de aplicaciones XCtest válido debería producir una salida similar a esta:

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

Para obtener más información, consulte Uso de XCTest para iOS y AWS Device Farm.

XCTEST_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find the executable value in the Info.plist file. Please unzip your test package and then open Info.plist file, verify that the key "CFBundleExecutable" is specified, and try again.

En el siguiente ejemplo, el nombre del paquete es swiftExampleTests.xctest-1.zip.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip swiftExampleTests.xctest-1.zip
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Debe encontrar el archivo *Info.plist* dentro de un directorio *.xctest*, como *swiftExampleTests.xctest* en nuestro ejemplo:

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

3. Para encontrar el valor del nombre del paquete, puede abrir Info.plist mediante Xcode o Python.

   Para Python, puede instalar el módulo biplist ejecutando el siguiente comando:

   ```
   $ pip install biplist
   ```

4. A continuación, abra Python y ejecute el siguiente comando:

   ```
   import biplist
   info_plist = biplist.readPlist('swiftExampleTests.xctest/Info.plist')
   print info_plist['CFBundleExecutable']
   ```

   Un paquete de aplicaciones XCtest válido debería producir una salida similar a esta:

```
swiftExampleTests
```

Para obtener más información, consulte Uso de XCTest para iOS y AWS Device Farm.

Solución de problemas de las pruebas de interfaz de usuario de XCTest en AWS Device Farm

En el siguiente tema se muestra una lista de mensajes de error que se producen durante la carga de las pruebas de XCTest UI y recomienda soluciones para resolver cada error.

> **Note**
>
> Las siguientes instrucciones se basan en Linux x86_64 y Mac.

XCTEST_UI_TEST_PACKAGE_UNZIP_FAILED

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not open your test IPA file. Please verify that the file is valid and try again.

Asegúrese de que puede descomprimir el paquete de aplicaciones sin errores. En el siguiente ejemplo, el nombre del paquete es swift-sample-UI.ipa.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

Un paquete de aplicaciones iOS válido debería producir una salida similar a esta:

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

Para obtener más información, consulte XCTest UI.

XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> 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.

En el siguiente ejemplo, el nombre del paquete es swift-sample-UI.ipa.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Si el paquete de XCTest UI es válido, encontrará el directorio *`Payload`* dentro del directorio de trabajo.

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

Para obtener más información, consulte XCTest UI.

XCTEST_UI_TEST_PACKAGE_APP_DIR_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> 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.

En el siguiente ejemplo, el nombre del paquete es swift-sample-UI.ipa.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Si el paquete de XCTest UI es válido, encontrará un directorio *`.app`*, como *`swift-sampleUITests-Runner.app`* en nuestro ejemplo, dentro del directorio *`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 obtener más información, consulte XCTest UI.

XCTEST_UI_TEST_PACKAGE_PLUGINS_DIR_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> 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.

En el siguiente ejemplo, el nombre del paquete es swift-sample-UI.ipa.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Si el paquete de XCTest UI es válido, encontrará el directorio *`Plugins`* dentro de un directorio *`.app`*. En nuestro ejemplo, el directorio se denomina *`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 obtener más información, consulte XCTest UI.

XCTEST_UI_TEST_PACKAGE_XCTEST_DIR_MISSING_IN_PLUGINS_DIR

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> 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.

En el siguiente ejemplo, el nombre del paquete es swift-sample-UI.ipa.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Si el paquete de XCTest UI es válido, encontrará un directorio *`.xctest`* dentro del directorio *`Plugins`*. En nuestro ejemplo, el directorio se denomina *`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 obtener más información, consulte XCTest UI.

XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> 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.

En el siguiente ejemplo, el nombre del paquete es swift-sample-UI.ipa.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Si el paquete de XCTest UI es válido, encontrará el archivo *Info.plist* dentro del directorio *.app*. En nuestro siguiente ejemplo, el directorio se denomina *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 obtener más información, consulte XCTest UI.

XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING_IN_XCTEST_DIR

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> 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.

En el siguiente ejemplo, el nombre del paquete es swift-sample-UI.ipa.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Si el paquete de XCTest UI es válido, encontrará el archivo *Info.plist* dentro del directorio de trabajo *.xctest*. En nuestro siguiente ejemplo, el directorio se denomina *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 obtener más información, consulte XCTest UI.

XCTEST_UI_TEST_PACKAGE_CPU_ARCHITECTURE_VALUE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> 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.

En el siguiente ejemplo, el nombre del paquete es swift-sample-UI.ipa.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Debe encontrar el archivo *Info.plist* dentro de un directorio *.app*, como *swift-sampleUITests-Runner.app* en nuestro ejemplo:

```
.
`-- 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 el valor de la arquitectura de la CPU, puede abrir Info.plist mediante Xcode o Python.

   Para Python, puede instalar el módulo biplist ejecutando el siguiente comando:

   ```
   $ pip install biplist
   ```

4. A continuación, abra Python y ejecute el siguiente comando:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/
   Info.plist')
   print info_plist['UIRequiredDeviceCapabilities']
   ```

   Un paquete de XCTest UI válido debería producir una salida similar a esta:

   ```
   ['armv7']
   ```

   Para obtener más información, consulte XCTest UI.

XCTEST_UI_TEST_PACKAGE_PLATFORM_VALUE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> 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.

En el siguiente ejemplo, el nombre del paquete es swift-sample-UI.ipa.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Debe encontrar el archivo *Info.plist* dentro de un directorio *.app*, como *swift-sampleUITests-Runner.app* en nuestro ejemplo:

   ```
   .
   `-- 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 el valor de la plataforma, puede abrir Info.plist mediante Xcode o Python.

   Para Python, puede instalar el módulo biplist ejecutando el siguiente comando:

   ```
   $ pip install biplist
   ```

4. A continuación, abra Python y ejecute el siguiente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['CFBundleSupportedPlatforms']
```

Un paquete de XCTest UI válido debería producir una salida similar a esta:

```
['iPhoneOS']
```

Para obtener más información, consulte XCTest UI.

XCTEST_UI_TEST_PACKAGE_WRONG_PLATFORM_DEVICE_VALUE

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> 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.

En el siguiente ejemplo, el nombre del paquete es swift-sample-UI.ipa.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Debe encontrar el archivo *Info.plist* dentro de un directorio *.app*, como *swift-sampleUITests-Runner.app* en nuestro ejemplo:

   ```
   .
   ```

```
`-- 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 el valor de la plataforma, puede abrir Info.plist mediante Xcode o Python.

   Para Python, puede instalar el módulo biplist ejecutando el siguiente comando:

   ```
   $ pip install biplist
   ```

4. A continuación, abra Python y ejecute el siguiente comando:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
   print info_plist['CFBundleSupportedPlatforms']
   ```

   Un paquete de XCTest UI válido debería producir una salida similar a esta:

   ```
   ['iPhoneOS']
   ```

   Si el paquete de XCTest UI es válida, el valor no debería contener la palabra clave `simulator`.

   Para obtener más información, consulte XCTest UI.

XCTEST_UI_TEST_PACKAGE_FORM_FACTOR_VALUE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> 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.

En el siguiente ejemplo, el nombre del paquete es swift-sample-UI.ipa.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip swift-sample-UI.ipa
```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

   Debe encontrar el archivo *Info.plist* dentro de un directorio *.app*, como *swift-sampleUITests-Runner.app* en nuestro ejemplo:

```
.
`-- 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 el valor del factor de forma, puede abrir Info.plist mediante Xcode o Python.

   Para Python, puede instalar el módulo biplist ejecutando el siguiente comando:

```
$ pip install biplist
```

4. A continuación, abra Python y ejecute el siguiente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['UIDeviceFamily']
```

   Un paquete de XCTest UI válido debería producir una salida similar a esta:

```
[1, 2]
```

Para obtener más información, consulte XCTest UI.

XCTEST_UI_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> We could not find the package ñame 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.

En el siguiente ejemplo, el nombre del paquete es swift-sample-UI.ipa.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip swift-sample-UI.ipa
```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

   Debe encontrar el archivo *Info.plist* dentro de un directorio *.app*, como *swift-sampleUITests-Runner.app* en nuestro ejemplo:

```
.
`-- 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 el valor del nombre del paquete, puede abrir Info.plist mediante Xcode o Python.

Para Python, puede instalar el módulo biplist ejecutando el siguiente comando:

```
$ pip install biplist
```

4. A continuación, abra Python y ejecute el siguiente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
print info_plist['CFBundleIdentifier']
```

Un paquete de XCTest UI válido debería producir una salida similar a esta:

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

Para obtener más información, consulte XCTest UI.

XCTEST_UI_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> 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.

En el siguiente ejemplo, el nombre del paquete es swift-sample-UI.ipa.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip swift-sample-UI.ipa
```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

```
$ tree .
```

Debe encontrar el archivo *Info.plist* dentro de un directorio *.app*, como *swift-sampleUITests-Runner.app* en nuestro ejemplo:

```
.
`-- 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 el valor del ejecutable puede abrir Info.plist mediante Xcode o Python.

   Para Python, puede instalar el módulo biplist ejecutando el siguiente comando:

   ```
   $ pip install biplist
   ```

4. A continuación, abra Python y ejecute el siguiente comando:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Info.plist')
   print info_plist['CFBundleExecutable']
   ```

   Un paquete de XCTest UI válido debería producir una salida similar a esta:

   ```
   XCTRunner
   ```

   Para obtener más información, consulte XCTest UI.

XCTEST_UI_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> 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.

En el siguiente ejemplo, el nombre del paquete es swift-sample-UI.ipa.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Debe encontrar el archivo *Info.plist* dentro de un directorio *.app*, como *swift-sampleUITests-Runner.app* en nuestro ejemplo:

   ```
   .
   `-- 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 el valor del nombre del paquete, puede abrir Info.plist mediante Xcode o Python.

   Para Python, puede instalar el módulo biplist ejecutando el siguiente comando:

   ```
   $ pip install biplist
   ```

4. A continuación, abra Python y ejecute el siguiente comando:

```
import biplist
info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Plugins/
swift-sampleUITests.xctest/Info.plist')
print info_plist['CFBundleIdentifier']
```

Un paquete de XCTest UI válido debería producir una salida similar a esta:

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

Para obtener más información, consulte XCTest UI.

XCTEST_UI_TEST_PACKAGE_TEST_EXECUTABLE_VALUE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

> **Warning**
>
> 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.

En el siguiente ejemplo, el nombre del paquete es swift-sample-UI.ipa.

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip swift-sample-UI.ipa
   ```

2. Después de descomprimir correctamente el paquete, ejecute el siguiente comando para que aparezca la estructura de árbol de directorios de trabajo:

   ```
   $ tree .
   ```

   Debe encontrar el archivo *`Info.plist`* dentro de un directorio *`.app`*, como *`swift-sampleUITests-Runner.app`* en nuestro ejemplo:

```
.
`-- 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 el valor del ejecutable puede abrir Info.plist mediante Xcode o Python.

   Para Python, puede instalar el módulo biplist ejecutando el siguiente comando:

   ```
   $ pip install biplist
   ```

4. A continuación, abra Python y ejecute el siguiente comando:

   ```
   import biplist
   info_plist = biplist.readPlist('Payload/swift-sampleUITests-Runner.app/Plugins/
   swift-sampleUITests.xctest/Info.plist')
   print info_plist['CFBundleExecutable']
   ```

   Un paquete de XCTest UI válido debería producir una salida similar a esta:

   ```
   swift-sampleUITests
   ```

   Para obtener más información, consulte XCTest UI.

Seguridad en AWS Device Farm

La seguridad en la nube de AWS es la mayor prioridad. Como cliente de AWS, se beneficia de una arquitectura de red y un centro de datos que se han diseñado para satisfacer los requisitos de seguridad de las organizaciones más exigentes.

La seguridad es una responsabilidad compartida entre AWS y usted. El modelo de responsabilidad compartida la describe como seguridad de la nube y seguridad en la nube:

- Seguridad de la nube: AWS es responsable de proteger la infraestructura que ejecuta los servicios de AWS en la nube de AWS. AWS también proporciona servicios que puede utilizar de forma segura. Los auditores externos prueban y verifican periódicamente la eficacia de nuestra seguridad como parte de los AWSProgramas de conformidad de . Para obtener información sobre los programas de conformidad que se aplican a AWS Device Farm, consulte Servicios de AWS en el ámbito del programa de conformidad.
- Seguridad en la nube: su responsabilidad se determina según el servicio de AWS que utilice. También es responsable de otros factores, incluida la confidencialidad de los datos, los requisitos de la empresa y la legislación y los reglamentos aplicables.

Esta documentación le ayuda a comprender cómo aplicar el modelo de responsabilidad compartida cuando se utiliza Device Farm. En los siguientes temas, se le mostrará cómo configurar Device Farm para satisfacer sus objetivos de seguridad y conformidad. También aprenderá a utilizar otros servicios de AWS que le ayudarán a monitorear y a proteger los recursos de Device Farm.

Temas

Administración de identidades y accesos en AWS Device Farm

Público

La forma en que utilice AWS Identity and Access Management (IAM) difiere en función del trabajo que realice en Device Farm.

Usuario de servicio: si utiliza el servicio de Device Farm para realizar su trabajo, su administrador le proporciona las credenciales y los permisos que necesita. A medida que utilice más características de Device Farm para realizar su trabajo, es posible que necesite permisos adicionales. Entender cómo se administra el acceso puede ayudarlo a solicitar los permisos correctos al administrador. Si no puede acceder a una característica en Device Farm, consulte Solución de problemas de identidad y acceso a AWS Device Farm.

Administrador de servicio: si está a cargo de los recursos de ACM en su empresa, es probable que tenga acceso completo a Device Farm. Su trabajo consiste en determinar a qué características y recursos de Device Farm deben acceder los usuarios del servicio. Luego, debe enviar solicitudes a su administrador de IAM para cambiar los permisos de los usuarios de su servicio. Revise la información de esta página para conocer los conceptos básicos de IAM. Para obtener más información sobre cómo su empresa puede utilizar IAM con Device Farm, consulte Cómo funciona AWS Device Farm con IAM.

Administrador de IAM: si es un administrador de IAM, es posible que quiera conocer más detalles sobre cómo escribir políticas para administrar el acceso a Device Farm. Para consultar ejemplos de políticas basadas en la identidad de Device Farm que puede utilizar en IAM, consulte Ejemplos de políticas basadas en identidad de AWS Device Farm.

Autenticación con identidades

La autenticación es la manera de iniciar sesión en AWS mediante credenciales de identidad. Debe estar autenticado (haber iniciado sesión en AWS) como Usuario raíz de la cuenta de AWS, como un usuario de IAM o asumiendo un rol de IAM.

Puede iniciar sesión en AWS como una identidad federada mediante las credenciales proporcionadas a través de una fuente de identidad de AWS IAM Identity Center. Los usuarios (del IAM Identity Center), la autenticación de inicio de sesión único de su empresa y sus credenciales de Google o Facebook son ejemplos de identidades federadas. Al iniciar sesión como una identidad federada, su administrador habrá configurado previamente la federación de identidades mediante

roles de IAM. Cuando accede a AWS mediante la federación, está asumiendo un rol de forma indirecta.

Según el tipo de usuario que sea, puede iniciar sesión en la AWS Management Console o en el portal de acceso a AWS. Para obtener más información sobre el inicio de sesión en AWS, consulte Cómo iniciar sesión en su Cuenta de AWS en la Guía del usuario de AWS Sign-In.

Si accede a AWS mediante programación, AWS proporciona un kit de desarrollo de software (SDK) y una interfaz de la línea de comandos (CLI) para firmar criptográficamente las solicitudes mediante el uso de las credenciales. Si no utiliza las herramientas de AWS, debe firmar usted mismo las solicitudes. Para obtener más información sobre la firma de solicitudes, consulte Firma de solicitudes API de AWS en la Guía del usuario de IAM.

Independientemente del método de autenticación que utilice, es posible que deba proporcionar información de seguridad adicional. Por ejemplo, AWS le recomienda el uso de la autenticación multifactor (MFA) para aumentar la seguridad de su cuenta. Para más información, consulte Autenticación multifactor en la Guía del usuario de AWS IAM Identity Center y Uso de la autenticación multifactor (MFA) en AWS en la Guía del usuario de IAM.

Usuario raíz de Cuenta de AWS

Cuando se crea una Cuenta de AWS, se comienza con una identidad de inicio de sesión que tiene acceso completo a todos los recursos y Servicios de AWS de la cuenta. Esta identidad recibe el nombre de usuario raíz de la Cuenta de AWS y se accede a ella iniciando sesión con el email y la contraseña que utilizó para crear la cuenta. Recomendamos encarecidamente que no utilice el usuario raíz para sus tareas diarias. Proteja las credenciales del usuario raíz y utilícelas solo para las tareas que solo el usuario raíz pueda realizar. Para ver la lista completa de las tareas que requieren que inicie sesión como usuario raíz, consulte Tareas que requieren credenciales de usuario raíz en la Guía del usuario de IAM.

Usuarios y grupos de IAM

Un usuario de IAM es una identidad en su Cuenta de AWS que dispone de permisos específicos para una sola persona o aplicación. Siempre que sea posible, recomendamos emplear credenciales temporales, en lugar de crear usuarios de IAM que tengan credenciales de larga duración como contraseñas y claves de acceso. No obstante, si tiene casos de uso específicos que requieran credenciales de larga duración con usuarios de IAM, recomendamos rotar las claves de acceso. Para más información, consulte Rotar las claves de acceso periódicamente para casos de uso que requieran credenciales de larga duración en la Guía del usuario de IAM.

Un grupo de IAM es una identidad que especifica un conjunto de usuarios de IAM. No puede iniciar sesión como grupo. Puede usar los grupos para especificar permisos para varios usuarios a la vez. Los grupos facilitan la administración de los permisos de grandes conjuntos de usuarios. Por ejemplo, podría tener un grupo cuyo nombre fuese IAMAdmins y conceder permisos a dicho grupo para administrar los recursos de IAM.

Los usuarios son diferentes de los roles. Un usuario se asocia exclusivamente a una persona o aplicación, pero la intención es que cualquier usuario pueda asumir un rol que necesite. Los usuarios tienen credenciales permanentes a largo plazo y los roles proporcionan credenciales temporales. Para más información, consulte Cuándo crear un usuario de IAM (en lugar de un rol) en la Guía del usuario de IAM.

Roles de IAM

Un rol de IAM es una identidad de tu Cuenta de AWS que dispone de permisos específicos. Es similar a un usuario de IAM, pero no está asociado a una determinada persona. Puede asumir temporalmente un rol de IAM en la AWS Management Console cambiando de roles. Puede asumir un rol llamando a una operación de AWS CLI o de la API de AWS, o utilizando una URL personalizada. Para más información sobre los métodos para el uso de roles, consulte Uso de roles de IAM en la Guía del usuario de IAM.

Los roles de IAM con credenciales temporales son útiles en las siguientes situaciones:

- Acceso de usuario federado: para asignar permisos a una identidad federada, puede crear un rol y definir sus permisos. Cuando se autentica una identidad federada, se asocia la identidad al rol y se le conceden los permisos define el rol. Para obtener información acerca de roles para federación, consulte Creación de un rol para un proveedor de identidades de terceros en la Guía del usuario de IAM. Si utiliza el IAM Identity Center, debe configurar un conjunto de permisos. El Centro de identidades de IAM correlaciona el conjunto de permisos con un rol en IAM para controlar a qué pueden acceder sus identidades después de autenticarse. Para obtener información acerca de los conjuntos de permisos, consulte Conjuntos de permisos en la Guía del usuario de AWS IAM Identity Center.
- Permisos de usuario de IAM temporales: un usuario de IAM puede asumir un rol de IAM para recibir temporalmente permisos distintos que le permitan realizar una tarea concreta.
- Acceso entre cuentas: puede utilizar un rol de IAM para permitir que alguien (una entidad principal de confianza) de otra cuenta acceda a los recursos de la cuenta. Los roles son la forma principal de conceder acceso entre cuentas. No obstante, con algunos Servicios de AWS se puede asociar una política directamente a un recurso (en lugar de utilizar un rol como representante). Para

obtener información acerca de la diferencia entre los roles y las políticas basadas en recursos para el acceso entre cuentas, consulte Cómo los roles de IAM difieren de las políticas basadas en recursos en la Guía del usuario de IAM.

- Acceso entre servicios: algunos Servicios de AWS utilizan características de otros Servicios de AWS. Por ejemplo, cuando realiza una llamada en un servicio, es común que ese servicio ejecute aplicaciones en Amazon EC2 o almacene objetos en Amazon S3. Es posible que un servicio haga esto usando los permisos de la entidad principal, usando un rol de servicio o usando un rol vinculado a servicios.
  - Reenviar sesiones de acceso (FAS): cuando utiliza un rol o un usuario de IAM para llevar a cabo acciones en AWS, se le considera una entidad principal. Cuando utiliza algunos servicios, es posible que realice una acción que desencadene otra acción en un servicio diferente. FAS utiliza los permisos de la entidad principal para llamar a un Servicio de AWS, combinados con el Servicio de AWS solicitante para realizar solicitudes a servicios posteriores. Las solicitudes de FAS solo se realizan cuando un servicio recibe una solicitud que requiere interacciones con otros Servicios de AWS o recursos para completarse. En este caso, debe tener permisos para realizar ambas acciones. Para obtener información sobre las políticas a la hora de realizar solicitudes de FAS, consulte Reenviar sesiones de acceso.
  - Rol de servicio: un rol de servicio es un rol de IAM que adopta un servicio para realizar acciones en su nombre. Un administrador de IAM puede crear, modificar y eliminar un rol de servicio desde IAM. Para más información, consulte Creación de un rol para delegar permisos a un Servicio de AWS en la Guía del usuario de IAM.
  - Rol vinculado al servicio: un rol vinculado al servicio es un tipo de rol de servicio que está vinculado a un Servicio de AWS. El servicio puede asumir el rol para realizar una acción en su nombre. Los roles vinculados a servicios aparecen en la Cuenta de AWS y son propiedad del servicio. Un administrador de IAM puede ver, pero no editar, los permisos de los roles vinculados a servicios.
- Aplicaciones que se ejecutan en Amazon EC2: puede utilizar un rol de IAM que le permita administrar credenciales temporales para las aplicaciones que se ejecutan en una instancia de EC2 y realizan solicitudes a la AWS CLI o a la API de AWS. Es preferible hacerlo de este modo a almacenar claves de acceso en la instancia EC2. Para asignar un rol de AWS a una instancia de EC2 y ponerla a disposición de todas las aplicaciones, cree un perfil de instancia asociado a la instancia. Un perfil de instancia contiene el rol y permite a los programas que se ejecutan en la instancia EC2 obtener credenciales temporales. Para más información, consulte Uso de un rol de IAM para conceder permisos a aplicaciones que se ejecutan en instancias Amazon EC2 en la Guía del usuario de IAM.

Para obtener información sobre el uso de los roles de IAM, consulte Cuándo crear un rol de IAM (en lugar de un usuario) en la Guía del usuario de IAM.

Cómo funciona AWS Device Farm con IAM

Antes de utilizar IAM para administrar el acceso a Device Farm, debe comprender qué características de IAM están disponibles para su uso con Device Farm. Para obtener una perspectiva general sobre cómo funcionan Device Farm y otros servicios de AWS con IAM, consulte AWSServicios que funcionan con IAM en la Guía del usuario de IAM.

Temas

- Políticas basadas en identidad de Device Farm
- Políticas basadas en recursos de Device Farm
- Listas de control de acceso
- Autorización basada en etiquetas de Device Farm
- Roles de IAM en Device Farm

Políticas basadas en identidad de Device Farm

Con las políticas basadas en identidades de IAM, puede especificar las acciones permitidas o denegadas, así como los recursos y las condiciones en las que se permiten o deniegan las acciones. Device Farm admite acciones, claves de condición y recursos específicos. Para obtener información sobre todos los elementos que utiliza en una política JSON, consulte Referencia de los elementos de las políticas JSON de IAM en la Guía del usuario de IAM.

Acciones

Los administradores pueden utilizar las políticas JSON de AWS para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Action` de una política JSON describe las acciones que puede utilizar para conceder o denegar el acceso en una política. Las acciones de la política generalmente tienen el mismo nombre que la operación de API AWS asociada. Hay algunas excepciones, como acciones de solo permiso que no tienen una operación de API coincidente. También hay algunas operaciones que requieren varias acciones en una política. Estas acciones adicionales se denominan acciones dependientes.

Incluya acciones en una política para conceder permisos y así llevar a cabo la operación asociada.

Las acciones de políticas de Device Farm utilizan el siguiente prefijo antes de la acción: `devicefarm:`. Por ejemplo, para conceder a alguien permiso para iniciar sesiones de Selenium con la operación de la API `CreateTestGridUrl` de Device Farm, incluya la acción `devicefarm:CreateTestGridUrl` en la política. Las instrucciones de la política deben incluir un elemento `Action` o un elemento `NotAction`. Device Farm define su propio conjunto de acciones que describen las tareas que se pueden realizar con este servicio.

Para especificar varias acciones en una única instrucción, sepárelas con comas del siguiente modo:

```
"Action": [
      "devicefarm:action1",
      "devicefarm:action2"
```

Puede utilizar caracteres comodín para especificar varias acciones (*). Por ejemplo, para especificar todas las acciones que comiencen con la palabra `List`, incluya la siguiente acción:

```
"Action": "devicefarm:List*"
```

Para ver una lista de las acciones de Device Farm, consulte Acciones definidas por AWS Device Farm en la Referencia de autorizaciones de servicio de IAM.

Recursos

Los administradores pueden utilizar las políticas JSON de AWS para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Resource` de la política JSON especifica el objeto u objetos a los que se aplica la acción. Las instrucciones deben contener un elemento `Resource` o `NotResource`. Como práctica recomendada, especifique un recurso utilizando el Nombre de recurso de Amazon (ARN). Puede hacerlo para acciones que admitan un tipo de recurso específico, conocido como permisos de nivel de recurso.

Para las acciones que no admiten permisos de nivel de recurso, como las operaciones de descripción, utilice un carácter comodín (*) para indicar que la instrucción se aplica a todos los recursos.

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

El recurso de instancia de Amazon EC2 tiene el siguiente ARN:

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

Para obtener más información acerca del formato de los ARN, consulte Nombres de recursos de Amazon (ARN) y espacios de nombres de servicios de AWS.

Por ejemplo, para especificar la instancia de `i-1234567890abcdef0` en su instrucción, utilice el siguiente ARN:

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:instance/i-1234567890abcdef0"
```

Para especificar todas las instancias que pertenecen a una cuenta, utilice el carácter comodín (*):

```
"Resource": "arn:aws:ec2:us-east-1:123456789012:instance/*"
```

Algunas acciones de Device Farm, como las que se utilizan para crear recursos, no se pueden llevar a cabo en un recurso. En dichos casos, debe utilizar el carácter comodín (*).

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

En muchas acciones de la API de Amazon EC2 se utilizan varios recursos. Por ejemplo, `AttachVolume` asocia un volumen de Amazon EBS a una instancia, por lo que un usuario de IAM debe tener permisos para utilizar el volumen y la instancia. Para especificar varios recursos en una única instrucción, separe los ARN con comas.

```
"Resource": [
      "resource1",
      "resource2"
```

Para ver una lista de los tipos de recursos de Device Farm y sus ARN, consulte Recursos definidos por AWS Device Farm en la Referencia de autorizaciones de servicio de IAM. Para saber con qué acciones puede especificar el ARN de cada recurso, consulte Acciones definidas por AWS Device Farm en la Referencia de autorizaciones de servicio de IAM.

Claves de condición

Los administradores pueden utilizar las políticas JSON de AWS para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

El elemento `Condition` (o bloque de `Condition`) permite especificar condiciones en las que entra en vigor una instrucción. El elemento `Condition` es opcional. Puede crear expresiones condicionales que utilicen operadores de condición, tales como igual o menor que, para que la condición de la política coincida con los valores de la solicitud.

Si especifica varios elementos de `Condition` en una instrucción o varias claves en un único elemento de `Condition`, AWS las evalúa mediante una operación lógica `AND`. Si especifica varios valores para una única clave de condición, AWS evalúa la condición con una operación `OR` lógica. Se deben cumplir todas las condiciones antes de que se concedan los permisos de la instrucción.

También puede utilizar variables de marcador de posición al especificar condiciones. Por ejemplo, puede conceder un permiso de usuario de IAM para acceder a un recurso solo si está etiquetado con su nombre de usuario de IAM. Para más información, consulte Elementos de la política de IAM: variables y etiquetas en la Guía del usuario de IAM.

AWS admite claves de condición globales y claves de condición específicas del servicio. Para ver todas las claves de condición globales de AWS, consulte Claves de contexto de condición globales de AWS en la Guía del usuario de IAM.

Device Farm define su propio conjunto de claves de condición y también admite el uso de algunas claves de condición globales. Para ver todas las claves de condición globales de AWS, consulte Claves de contexto de condición globales de AWS en la Guía del usuario de IAM.

Para ver una lista de las claves de condición de Device Farm, consulte Claves de condición para AWS Device Farm en la Referencia de autorizaciones de servicio de IAM. Para saber con qué acciones y recursos puede usar una clave de condición, consulte Acciones definidas por AWS Device Farm en la Referencia de autorizaciones de servicio de IAM.

Ejemplos

Para ver ejemplos de políticas basadas en identidad de Device Farm, consulte Ejemplos de políticas basadas en identidad de AWS Device Farm.

Políticas basadas en recursos de Device Farm

Device Farm no admite las políticas basadas en recursos.

Listas de control de acceso

Device Farm no admite las listas de control de acceso (ACL).

Autorización basada en etiquetas de Device Farm

Puede adjuntar etiquetas a los recursos de Device Farm o transferirlas en una solicitud a Device Farm. Para controlar el acceso en función de etiquetas, debe proporcionar información de las etiquetas en el elemento de condición de una política utilizando las claves de condición `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` o `aws:TagKeys`. Para obtener más información acerca del etiquetado de recursos de Device Farm, consulte Etiquetado en Device Farm.

Para consultar un ejemplo de política basada en la identidad para limitar el acceso a un recurso en función de las etiquetas de ese recurso, consulte Visualización de proyectos de pruebas de navegadores de escritorio de Device Farm basados en etiquetas.

Roles de IAM en Device Farm

Un rol de IAM es una entidad de la cuenta de AWS que dispone de permisos específicos.

Uso de credenciales temporales con Device Farm

Device Farm admite el uso de credenciales temporales.

Puede utilizar credenciales temporales para iniciar sesión con federación para asumir un rol de IAM o un rol de acceso entre cuentas. Las credenciales de seguridad temporales se obtienen mediante una llamada a operaciones de la API de AWS STS, como AssumeRole o GetFederationToken.

Roles vinculados al servicio

Los roles vinculados a servicios permiten a los servicios de AWS obtener acceso a los recursos de otros servicios para completar una acción en su nombre. Los roles vinculados a servicios aparecen en la cuenta de IAM y son propiedad del servicio. Un administrador de IAM puede ver, pero no editar, los permisos de los roles vinculados a servicios.

Device Farm utiliza funciones vinculadas a servicios en la característica de pruebas del navegador de escritorio Device Farm. Para obtener información sobre estas funciones, consulte Uso de funciones vinculadas a servicios en las pruebas del navegador de escritorio de Device Farm en la guía para desarrolladores.

Roles de servicio

Device Farm no admite roles de servicio.

Esta característica permite que un servicio asuma un rol de servicio en su nombre. Este rol permite que el servicio obtenga acceso a los recursos de otros servicios para completar una acción en su

nombre. Los roles de servicio aparecen en su cuenta de IAM y son propiedad de la cuenta. Esto significa que un administrador de IAM puede cambiar los permisos de este rol. Sin embargo, hacerlo podría deteriorar la funcionalidad del servicio.

Administración de acceso mediante políticas

Para controlar el acceso en AWS, se crean políticas y se adjuntan a identidades o recursos de AWS. Una política es un objeto de AWS que, cuando se asocia a una identidad o un recurso, define sus permisos. AWS evalúa estas políticas cuando una entidad principal (sesión de rol, usuario o usuario raíz) realiza una solicitud. Los permisos en las políticas determinan si la solicitud se permite o se deniega. La mayoría de las políticas se almacenan en AWS como documentos JSON. Para obtener más información sobre la estructura y el contenido de los documentos de política JSON, consulte Información general de políticas JSON en la Guía del usuario de IAM.

Los administradores pueden utilizar las políticas JSON de AWS para especificar quién tiene acceso a qué. Es decir, qué entidad principal puede realizar acciones en qué recursos y en qué condiciones.

De forma predeterminada, los usuarios y los roles no tienen permisos. Para conceder permiso a los usuarios para realizar acciones en los recursos que necesiten, un administrador puede crear políticas de IAM. A continuación, el administrador puede añadir las políticas de IAM a roles y los usuarios pueden asumirlos.

Las políticas de IAM definen permisos para una acción independientemente del método que se utilice para realizar la operación. Por ejemplo, suponga que dispone de una política que permite la acción `iam:GetRole`. Un usuario con dicha política puede obtener información del rol de la AWS Management Console, la AWS CLI o la API de AWS.

Políticas basadas en identidades

Las políticas basadas en identidad son documentos de políticas de permisos JSON que puede asociar a una identidad, como un usuario de IAM, un grupo de usuarios o un rol. Estas políticas controlan qué acciones pueden realizar los usuarios y los roles, en qué recursos y en qué condiciones. Para obtener más información sobre cómo crear una política basada en identidad, consulte Creación de políticas de IAM en la Guía del usuario de IAM.

Las políticas basadas en identidades pueden clasificarse además como políticas insertadas o políticas administradas. Las políticas insertadas se integran directamente en un único usuario, grupo o rol. Las políticas administradas son políticas independientes que puede asociar a varios usuarios,

grupos y roles de su Cuenta de AWS. Las políticas administradas incluyen las políticas administradas de AWS y las políticas administradas por el cliente. Para más información sobre cómo elegir una política administrada o una política insertada, consulte Elegir entre políticas administradas y políticas insertadas en la Guía del usuario de IAM.

En la siguiente tabla se exponen las políticas administradas de AWS de Device Farm.

Cambio	Descripción	Fecha
AWSDeviceFarmFullAccess	Proporciona acceso completo a todas las operaciones de AWS Device Farm.	15 de julio de 2015
AWSServiceRoleForDeviceFarmTestGrid	Permite que Device Farm acceda a los recursos de AWS en su nombre.	20 de mayo de 2021

Otros tipos de políticas

AWS admite otros tipos de políticas adicionales menos frecuentes. Estos tipos de políticas pueden establecer el máximo de permisos que los tipos de políticas más frecuentes le conceden.

- Límites de permisos: un límite de permisos es una característica avanzada que le permite establecer los permisos máximos que una política basada en identidad puede conceder a una entidad de IAM (usuario o rol de IAM). Puede establecer un límite de permisos para una entidad. Los permisos resultantes son la intersección de las políticas basadas en la identidad de la entidad y los límites de permisos. Las políticas basadas en recursos que especifique el usuario o rol en el campo `Principal` no estarán restringidas por el límite de permisos. Una denegación explícita en cualquiera de estas políticas anulará el permiso. Para obtener más información sobre los límites de los permisos, consulte Límites de permisos para las entidades de IAM en la Guía del usuario de IAM.
- Políticas de control de servicio (SCP): las SCP son políticas de JSON que especifican los permisos máximos de una empresa o una unidad organizativa en AWS Organizations. AWS Organizations es un servicio que le permite agrupar y administrar de manera centralizada varias Cuentas de AWS que posea su empresa. Si habilita todas las características en una empresa, entonces podrá aplicar políticas de control de servicio (SCP) a una o todas sus cuentas. Una SCP limita los permisos para las entidades de las cuentas de miembros, incluido cada Usuario raíz de la cuenta

de AWS. Para más información sobre Organizations y las SCP, consulte Funcionamiento de las SCP en la Guía del usuario de AWS Organizations.

- Políticas de sesión: las políticas de sesión son políticas avanzadas que se pasan como parámetro cuando se crea una sesión temporal mediante programación para un rol o un usuario federado. Los permisos de la sesión resultantes son la intersección de las políticas basadas en identidades del rol y las políticas de la sesión. Los permisos también pueden proceder de una política en función de recursos. Una denegación explícita en cualquiera de estas políticas anulará el permiso. Para más información, consulte Políticas de sesión en la Guía del usuario de IAM.

Varios tipos de políticas

Cuando se aplican varios tipos de políticas a una solicitud, los permisos resultantes son más complicados de entender. Para obtener información acerca de cómo AWS decide si permitir o no una solicitud cuando hay varios tipos de políticas implicados, consulte Lógica de evaluación de políticas en la Guía del usuario de IAM.

Ejemplos de políticas basadas en identidad de AWS Device Farm

De forma predeterminada, los usuarios y los roles de IAM no tienen permiso para crear, ver ni modificar recursos de Device Farm. Tampoco pueden realizar tareas mediante la AWS Management Console, la AWS CLI, o la API de AWS. Un administrador de IAM debe crear políticas de IAM que concedan permisos a los usuarios y a los roles para realizar operaciones de la API concretas en los recursos especificados que necesiten. El administrador debe adjuntar esas políticas a los usuarios o grupos de IAM que necesiten esos permisos.

Para obtener información acerca de cómo crear una política basada en identidad de IAM con estos documentos de políticas JSON de ejemplo, consulte Creación de políticas en la pestaña JSON en la Guía del usuario de IAM.

Temas

- Prácticas recomendadas relativas a políticas
- Cómo permitir a los usuarios consultar sus propios permisos
- Acceso a un proyecto de prueba de explorador de escritorio de Device Farm
- Visualización de proyectos de pruebas de navegadores de escritorio de Device Farm basados en etiquetas

Prácticas recomendadas relativas a políticas

Las políticas basadas en identidades determinan si alguien puede crear, acceder o eliminar los recursos de Device Farm de la cuenta. Estas acciones pueden generar costes adicionales para su Cuenta de AWS. Siga estas directrices y recomendaciones al crear o editar políticas basadas en identidades:

- Comience con las políticas administradas de AWS y continúe con los permisos de privilegio mínimo: a fin de comenzar a conceder permisos a los usuarios y las cargas de trabajo, utilice las políticas administradas de AWS, que conceden permisos para muchos casos de uso comunes. Están disponibles en su Cuenta de AWS. Se recomienda definir políticas administradas por el cliente de AWS específicas para los casos de uso a fin de reducir aún más los permisos. Con el fin de obtener más información, consulte las políticas administradas por AWS o las políticas administradas por AWS para funciones de trabajo en la Guía del usuario de IAM.
- Aplique permisos de privilegio mínimo: cuando establezca permisos con políticas de IAM, conceda solo los permisos necesarios para realizar una tarea. Para ello, debe definir las acciones que se pueden llevar a cabo en determinados recursos en condiciones específicas, también conocidos como permisos de privilegios mínimos. Con el fin de obtener más información sobre el uso de IAM para aplicar permisos, consulte Políticas y permisos en IAM en la Guía del usuario de IAM.
- Utilice condiciones en las políticas de IAM para restringir aún más el acceso: puede agregar una condición a sus políticas para limitar el acceso a las acciones y los recursos. Por ejemplo, puede escribir una condición de políticas para especificar que todas las solicitudes deben enviarse utilizando SSL. También puede usar condiciones para conceder acceso a acciones de servicios si se emplean a través de un Servicio de AWS determinado, como por ejemplo AWS CloudFormation. Para más información, consulte Elementos de la política JSON de IAM: condición en la Guía del usuario de IAM.
- Utilice el analizador de acceso de IAM para validar las políticas de IAM con el fin de garantizar la seguridad y funcionalidad de los permisos: el analizador de acceso de IAM valida políticas nuevas y existentes para que respeten el lenguaje (JSON) de las políticas de IAM y las prácticas recomendadas de IAM. El analizador de acceso de IAM proporciona más de 100 verificaciones de políticas y recomendaciones procesables para ayudar a crear políticas seguras y funcionales. Para más información, consulte la política de validación del Analizador de acceso de IAM en la Guía del usuario de IAM.
- Solicite la autenticación multifactor (MFA): si se encuentra en una situación en la que necesita usuarios raíz o de IAM en su Cuenta de AWS, active la MFA para mayor seguridad. Para solicitar la MFA cuando se invocan las operaciones de la API, agregue las condiciones de la MFA a sus

políticas. Para más información, consulte Configuración de acceso a una API protegida por MFA en la Guía del usuario de IAM.

Para obtener más información sobre las prácticas recomendadas de IAM, consulte las Prácticas recomendadas de seguridad en IAM en la Guía del usuario de IAM.

Cómo permitir a los usuarios consultar sus propios permisos

En este ejemplo, se muestra cómo podría crear una política que permita a los usuarios de IAM ver las políticas administradas e insertadas que se asocian a la identidad de sus usuarios. Esta política incluye permisos para realizar esta acción en la consola o mediante programación con la AWS CLI o la API de 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": "*"
```

```
        }
    ]
}
```

Acceso a un proyecto de prueba de explorador de escritorio de Device Farm

En este ejemplo quiere otorgar acceso a un usuario de IAM de su cuenta AWS a uno de sus proyectos de prueba de navegador de escritorio de Device Farm, `arn:aws:devicefarm:us-west-2:111122223333:testgrid-project:123e4567-e89b-12d3-a456-426655441111`. Desea que la cuenta pueda ver elementos relacionados con el proyecto.

Además del punto de enlace `devicefarm:GetTestGridProject`, la cuenta debe tener los puntos de enlace `devicefarm:ListTestGridSessions`, `devicefarm:GetTestGridSession`, `devicefarm:ListTestGridSessionActions` y `devicefarm:ListTestGridSessionArtifacts`.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"GetTestGridProject",
         "Effect":"Allow",
         "Action":[
            "devicefarm:GetTestGridProject"
         ],
         "Resource":"arn:aws:devicefarm:us-west-2:111122223333:testgrid-
project:123e4567-e89b-12d3-a456-426655441111"
      },
      {
         "Sid":"ViewProjectInfo",
         "Effect":"Allow",
         "Action":[
            "devicefarm:ListTestGridSessions",
            "devicefarm:ListTestGridSessionActions",
            "devicefarm:ListTestGridSessionArtifacts"
         ],
         "Resource":"arn:aws:devicefarm:us-west-2:111122223333:testgrid-*:123e4567-
e89b-12d3-a456-426655441111/*"
      }
   ]
}
```

Si utiliza sistemas de CI, debe proporcionar a cada corredor de CI credenciales de acceso únicas. Por ejemplo, es poco probable que un sistema de CI necesite más permisos que `devicefarm:ScheduleRun` o `devicefarm:CreateUpload`. La siguiente política de IAM describe una política mínima para permitir que un programa de ejecución de CI comience una prueba de aplicación nativa de Device Farm creando una carga y utilizándola para programar una ejecución de prueba:

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

Visualización de proyectos de pruebas de navegadores de escritorio de Device Farm basados en etiquetas

Puede utilizar las condiciones de su política basada en la identidad para controlar el acceso a los recursos de Device Farm basados en etiquetas. En este ejemplo se muestra cómo crear una política que permita la visualización de proyectos y sesiones. Se concede permiso si la etiqueta `Owner` del recurso solicitado coincide con el nombre de usuario de la cuenta solicitante.

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

```
              "devicefarm:ListTestGridSession*",
              "devicefarm:GetTestGridSession",
              "devicefarm:ListTestGridProjects"
              ],
            "Resource": [
               "arn:aws:devicefarm:us-west-2:testgrid-project:*/*"
               "arn:aws:devicefarm:us-west-2:testgrid-session:*/*"
               ],
            "Condition": {
               "StringEquals": {"aws:TagKey/Owner":"${aws:username}"}
            }
        }
    ]
}
```

También puede asociar esta política al usuario de IAM en su cuenta. Si un usuario denominado `richard-roe` intenta ver un proyecto o sesión de Device Farm, el proyecto debe tener la etiqueta `Owner=richard-roe` o `owner=richard-roe`. De lo contrario, se deniega el acceso al usuario. La clave de la etiqueta de condición `Owner` coincide con los nombres de las claves de condición `Owner` y `owner` porque no distinguen entre mayúsculas y minúsculas. Para obtener más información, consulte Elementos de la política de JSON de IAM: Condición en la Guía del usuario de IAM.

Solución de problemas de identidad y acceso a AWS Device Farm

Utilice la siguiente información para diagnosticar y solucionar los problemas comunes que puedan surgir cuando trabaje con Device Farm e IAM.

No tengo autorización para realizar una acción en Device Farm

Si recibe un error en la AWS Management Console que indica que no está autorizado para llevar a cabo la acción, debe ponerse en contacto con su administrador para recibir ayuda. Su administrador es la persona que le facilitó su nombre de usuario y contraseña.

En el siguiente ejemplo, el error se produce cuando el usuario de IAM, `mateojackson`, intenta utilizar la consola para ver detalles sobre una ejecución, pero no tiene permisos de `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
```

En este caso, Mateo pide a su administrador que actualice sus políticas de forma que pueda obtener acceso al `devicefarm:GetRun` del recurso `arn:aws:devicefarm:us-west-2:123456789101:run:123e4567-e89b-12d3-a456-426655440000/123e4567-e89b-12d3-a456-426655441111` mediante la acción `devicefarm:GetRun`.

No estoy autorizado a realizar actividades como: PassRole

Si recibe un error que indica que no tiene autorización para realizar la acción `iam:PassRole`, se deben actualizar las políticas para permitirle pasar un rol a Device Farm.

Algunos Servicios de AWS le permiten transferir un rol existente a dicho servicio en lugar de crear un nuevo rol de servicio o uno vinculado al servicio. Para ello, debe tener permisos para transferir el rol al servicio.

En el siguiente ejemplo, el error se produce cuando un usuario de IAM denominado `marymajor` intenta utilizar la consola para realizar una acción en Device Farm. Sin embargo, la acción requiere que el servicio cuente con permisos que concede un rol de servicio. Mary no tiene permisos para transferir el rol al servicio.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

En este caso, las políticas de Mary se deben actualizar para permitirle realizar la acción `iam:PassRole`.

Si necesita ayuda, póngase en contacto con su administrador de AWS. El administrador es la persona que le proporcionó las credenciales de inicio de sesión.

Quiero ver mis claves de acceso

Después de crear sus claves de acceso de usuario de IAM, puede ver su ID de clave de acceso en cualquier momento. Sin embargo, no puede volver a ver su clave de acceso secreta. Si pierde la clave de acceso secreta, debe crear un nuevo par de claves de acceso.

Las claves de acceso se componen de dos partes: un ID de clave de acceso (por ejemplo, `AKIAIOSFODNN7EXAMPLE`) y una clave de acceso secreta (por ejemplo, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). El ID de clave de acceso y la clave de acceso secreta se utilizan juntos, como un nombre de usuario y contraseña, para autenticar sus solicitudes. Administre sus claves de acceso con el mismo nivel de seguridad que para el nombre de usuario y la contraseña.

> **Important**
>
> No proporcione las claves de acceso a terceros, ni siquiera para que lo ayuden a buscar el ID de usuario canónico. Si lo hace, podría conceder a otra persona acceso permanente a su Cuenta de AWS.

Cuando crea un par de claves de acceso, se le pide que guarde el ID de clave de acceso y la clave de acceso secreta en un lugar seguro. La clave de acceso secreta solo está disponible en el momento de su creación. Si pierde la clave de acceso secreta, debe agregar nuevas claves de acceso a su usuario de IAM. Puede tener un máximo de dos claves de acceso. Si ya cuenta con dos, debe eliminar un par de claves antes de crear una nueva. Para consultar las instrucciones, consulte Administración de claves de acceso en la Guía del usuario de IAM.

Soy administrador y deseo permitir que otros obtengan acceso a Device Farm

Para permitir que otros obtengan acceso a Device Farm, debe crear una entidad de IAM (usuario o rol) para la persona o la aplicación que necesita acceso. Esta persona utilizará las credenciales de la entidad para acceder a AWS. A continuación, debe asociar una política a la entidad que le conceda los permisos correctos en Device Farm.

Para comenzar de inmediato, consulte Creación del primer grupo y usuario delegado de IAM en la Guía del usuario de IAM.

Deseo permitir que personas ajenas a mi cuenta de AWS puedan acceder a mis recursos de Device Farm

Puede crear un rol que los usuarios de otras cuentas o las personas externas a la organización puedan utilizar para acceder a sus recursos. Puede especificar una persona de confianza para que asuma el rol. En el caso de los servicios que admitan las políticas basadas en recursos o las listas de control de acceso (ACL), puede utilizar dichas políticas para conceder a las personas acceso a sus recursos.

Para más información, consulte lo siguiente:

- Para obtener información acerca de si Device Farm admite estas características, consulte Cómo funciona AWS Device Farm con IAM.

- Para obtener información acerca de cómo proporcionar acceso a los recursos de las Cuentas de AWS de su propiedad, consulte Proporcionar acceso a un usuario de IAM a otra Cuenta de AWS de la que es propietario en la Guía del usuario de IAM.
- Para obtener información acerca de cómo proporcionar acceso a los recursos a Cuentas de AWS de terceros, consulte Proporcionar acceso a Cuentas de AWS que son propiedad de terceros en la Guía del usuario de IAM.
- Para obtener información sobre cómo proporcionar acceso mediante una federación de identidades, consulte Proporcionar acceso a usuarios autenticados externamente (federación de identidades) en la Guía del usuario de IAM.
- Para obtener información sobre la diferencia entre los roles y las políticas basadas en recursos para el acceso entre cuentas, consulte Cómo los roles de IAM difieren de las políticas basadas en recursos en la Guía del usuario de IAM.

Validación de la conformidad en AWS Device Farm

Auditores externos evalúan la seguridad y la conformidad de AWS Device Farm como parte de varios programas de conformidad de AWS. Estos incluyen SOC, PCI, FedRAMP, HIPAA y otros. AWS Device Farm no está al alcance de ningún programa de cumplimiento de AWS.

Para obtener una lista de servicios de AWS en el ámbito de programas de conformidad específicos, consulte Servicios de AWS en el ámbito del programa de conformidad. Para obtener información general, consulte Programas de conformidad de AWS.

Puede descargar los informes de auditoría de terceros utilizando AWS Artifact. Para obtener más información, consulte Descarga de informes en AWS Artifact.

Su responsabilidad en el ámbito de la conformidad al usar Device Farm viene determinada por la confidencialidad de los datos, los objetivos de conformidad de su empresa y las leyes y regulaciones aplicables. AWS proporciona los siguientes recursos para ayudarlo con los requisitos de conformidad:

- Guías de inicio rápido de seguridad y conformidad: estas guías de implementación tratan consideraciones sobre arquitectura y ofrecen pasos para implementar los entornos de referencia centrados en la seguridad y la conformidad en AWS.
- Recursos de conformidad de AWS: este conjunto de manuales y guías podría aplicarse a su sector y ubicación.

- Evaluación de recursos con reglas en la Guía para desarrolladores de AWS Config: AWS Config evalúa en qué medida las configuraciones de sus recursos cumplen las prácticas internas, las directrices del sector y las normativas.
- AWS Security Hub: este servicio de AWS proporciona una vista integral de su estado de seguridad en AWS que lo ayuda a verificar la conformidad con los estándares y las prácticas recomendadas del sector de seguridad.

Protección de los datos en AWS Device Farm

El AWS modelo de responsabilidad compartida se aplica a la protección de datos en AWS Device Farm (Device Farm). Como se describe en este modelo, AWS es responsable de proteger la infraestructura global que ejecuta la totalidad de Nube de AWS. Usted es responsable de mantener el control sobre el contenido alojado en esta infraestructura. Usted también es responsable de las tareas de administración y configuración de seguridad para los Servicios de AWS que utiliza. Para obtener más información sobre la privacidad de los datos, consulte las Preguntas frecuentes sobre la privacidad de datos. Para obtener información sobre la protección de datos en Europa, consulte la publicación de blog AWS Shared Responsibility Model and GDPR en el Blog de seguridad de AWS.

Con fines de protección de datos, recomendamos proteger las credenciales de la cuenta de Cuenta de AWS y configurar cuentas de usuario individuales con AWS IAM Identity Center o AWS Identity and Access Management (IAM). De esta manera, solo se otorgan a cada usuario los permisos necesarios para cumplir sus obligaciones laborales. También recomendamos proteger sus datos de la siguiente manera:

- Utilice autenticación multifactor (MFA) en cada cuenta.
- Utilice SSL/TLS para comunicarse con los recursos de AWS. Se recomienda el uso de TLS 1.2 y recomendamos TLS 1.3.
- Configure la API y el registro de actividad del usuario con AWS CloudTrail.
- Utilice las soluciones de cifrado de AWS, junto con todos los controles de seguridad predeterminados dentro de los Servicios de AWS.
- Utilice servicios de seguridad administrados avanzados, como Amazon Macie, que lo ayuden a detectar y proteger los datos confidenciales almacenados en Amazon S3.
- Si necesita módulos criptográficos validados FIPS 140-2 al acceder a AWS a través de una interfaz de la línea de comandos o una API, utilice un punto de conexión de FIPS. Para obtener más información sobre los puntos de conexión de FIPS disponibles, consulte Estándar de procesamiento de la información federal (FIPS) 140-2.

Se recomienda encarecidamente no introducir nunca información confidencial o sensible, como, por ejemplo, direcciones de correo electrónico de clientes, en etiquetas o campos de formato libre, tales como el campo Nombre. Esto incluye los casos en los que debe trabajar con Device Farm u otros Servicios de AWS a través de la consola, la API, AWS CLI o los SDK AWS. Cualquier dato que ingrese en etiquetas o campos de formato libre utilizados para nombres se puede emplear para los registros de facturación o diagnóstico. Si proporciona una URL a un servidor externo, recomendamos encarecidamente que no incluya información de credenciales en la URL a fin de validar la solicitud para ese servidor.

Cifrado en tránsito

Los puntos de conexión de Device Farm solo admiten solicitudes HTTPS (SSL/TLS) firmadas, excepto cuando se indique lo contrario. Todo el contenido recuperado o colocado en Amazon S3 mediante la URL de carga se cifra mediante SSL/TLS. Para obtener más información sobre cómo las solicitudes HTTPS han iniciado sesión en AWS, consulte Firma de solicitudes de la API de AWS en la referencia general de AWS.

Es responsabilidad suya cifrar y proteger cualquier comunicación que realicen sus aplicaciones probadas, así como cualquier aplicación instalada en el proceso de ejecución de pruebas en el dispositivo.

Cifrado en reposo

La característica de prueba del navegador de escritorio de Device Farm admite el cifrado en reposo de los artefactos generados durante las pruebas.

Los datos de prueba de dispositivos móviles físicos de Device Farm no están cifrados en reposo.

Retención de datos

Los datos en Device Farm se conservan durante un tiempo limitado. Después de que caduque el período de retención, se eliminan los datos del almacenamiento de respaldo de Device Farm, pero los metadatos (ARN, fechas de carga, nombres de archivo, etc.) se conservan para su uso futuro. En la tabla siguiente se muestra el período de retención de varios tipos de contenido.

Tipo de contenido	Periodo de retención (días)
Aplicaciones cargadas	30

Tipo de contenido	Periodo de retención (días)
Paquetes de prueba cargados	30
Registros	400
Grabaciones de vídeo y otros artefactos	400

Es su responsabilidad archivar cualquier contenido que desee conservar durante períodos más largos.

Administración de datos

Los datos en Device Farm se administran de manera diferente según las características que se utilicen. En esta sección se explica cómo se administran los datos mientras se utiliza Device Farm y después de usarlo.

Pruebas del explorador de escritorio

Las instancias utilizadas durante las sesiones de Selenium no se guardan. Todos los datos generados como resultado de las interacciones del navegador se descartan cuando finaliza la sesión.

Actualmente, esta característica admite el cifrado en reposo para los artefactos generados durante la prueba.

Pruebas de dispositivos físicos

En las secciones siguientes se proporciona información acerca de los pasos que AWS sigue para limpiar o destruir los dispositivos después de haber utilizado Device Farm.

Los datos de prueba de dispositivos móviles físicos de Device Farm no están cifrados en reposo.

Flotas de dispositivos públicos

Una vez completada la ejecución de prueba, Device Farm realiza una serie de tareas de limpieza en cada dispositivo de la flota de dispositivos públicos, incluida la desinstalación de la aplicación. Si no podemos verificar la desinstalación de la aplicación o de cualquiera de los demás pasos de limpieza, el dispositivo se restablece a sus valores de fábrica antes de volver a ponerlo en uso.

> **Note**
>
> Es posible que, en algunos casos, los datos persistan de una sesión a otra, especialmente si utiliza el sistema de dispositivos de fuera del contexto de la aplicación. Por este motivo, y dado que Device Farm captura vídeo y registros de la actividad que se produce durante el uso de cada dispositivo, recomendamos que evite escribir información confidencial (por ejemplo, una cuenta de Google o un ID de Apple), datos personales y otros detalles confidenciales de seguridad durante la prueba automatizada y las sesiones de acceso remoto.

Dispositivos privados

Tras el vencimiento o la resolución del contrato de dispositivo privado, el dispositivo se deja de usar y se destruye de forma segura de conformidad con las políticas de destrucción de AWS. Para obtener más información, consulte Trabajar con dispositivos privados en AWS Device Farm.

Administración de claves

Actualmente, Device Farm no ofrece ninguna gestión de claves externa para el cifrado de datos, en reposo o en tránsito.

Privacidad del tráfico entre redes

Device Farm se puede configurar solo para dispositivos privados, para usar puntos de conexión de VPC de Amazon para conectarse a sus recursos en AWS. El acceso a cualquier infraestructura de AWS no pública asociada a su cuenta (por ejemplo, instancias de EC2 de Amazon sin una dirección IP pública) debe utilizar un punto de conexión de VPC. Independientemente de la configuración del punto de conexión de VPC, Device Farm aísla el tráfico de otros usuarios de la red de Device Farm.

No se garantiza que sus conexiones fuera de la red de AWS sean seguras o estén protegidas, y es responsabilidad suya proteger cualquier conexión a internet que realicen sus aplicaciones.

Resiliencia en AWS Device Farm

La infraestructura global de AWS se compone de regiones y zonas de disponibilidad de AWS. AWS Las regiones proporcionan varias zonas de disponibilidad físicamente independientes y aisladas que

se encuentran conectadas mediante redes con un alto nivel de rendimiento y redundancia, además de baja latencia. Con las zonas de disponibilidad, puede diseñar y utilizar aplicaciones y bases de datos que realizan una conmutación por error automática entre las zonas sin interrupciones. Las zonas de disponibilidad tienen una mayor disponibilidad, tolerancia a errores y escalabilidad que las infraestructuras tradicionales de centros de datos únicos o múltiples.

Para obtener más información sobre las regiones y zonas de disponibilidad de AWS, consulte Infraestructura global de AWS.

Dado que Device Farm solo está disponible en la región de `us-west-2`, recomendamos encarecidamente que implemente procesos de copia de seguridad y recuperación. Device Farm no debe ser la única fuente de contenido cargado.

Device Farm no garantiza la disponibilidad de dispositivos públicos. Estos dispositivos se toman dentro y fuera del grupo de dispositivos públicos en función de varios factores, como la tasa de fallos y el estado de cuarentena. No recomendamos que dependa de la disponibilidad de un dispositivo en el grupo de dispositivos público.

Seguridad de la infraestructura en AWS Device Farm

Como se trata de un servicio administrado, AWS Device Farm está protegido por la seguridad de red global de AWS. Para obtener información sobre los servicios de seguridad de AWS y cómo AWS protege la infraestructura, consulte Seguridad en la nube de AWS. Para diseñar su entorno de AWS con las prácticas recomendadas de seguridad de infraestructura, consulte Protección de la infraestructura en Portal de seguridad de AWS Well-Architected Framework.

Puede utilizar llamadas a la API publicadas en AWS para obtener acceso a Device Farm a través de la red. Los clientes deben admitir lo siguiente:

- Seguridad de la capa de transporte (TLS). Nosotros exigimos TLS 1.2 y recomendamos TLS 1.3.
- Conjuntos de cifrado con confidencialidad directa total (PFS) tales como DHE (Ephemeral Diffie-Hellman) o ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). La mayoría de los sistemas modernos como Java 7 y posteriores son compatibles con estos modos.

Además, las solicitudes deben estar firmadas mediante un ID de clave de acceso y una clave de acceso secreta que esté asociada a una entidad de seguridad de IAM. También puede utilizar AWS Security Token Service (AWS STS) para generar credenciales de seguridad temporales para firmar solicitudes.

Seguridad de la infraestructura para pruebas de dispositivos físicos

Los dispositivos se separan físicamente durante las pruebas de dispositivos físicos. El aislamiento de la red impide la comunicación entre dispositivos a través de redes inalámbricas.

Los dispositivos públicos se comparten y Device Farm hace un gran esfuerzo para garantizar la seguridad de los dispositivos a lo largo del tiempo. Algunas acciones, como los intentos de adquirir derechos de administrador completos en un dispositivo (práctica denominada rooting o jailbreaking), hacen que los dispositivos públicos se pongan en cuarentena. Se eliminan automáticamente del grupo público y se colocan en revisión manual.

Solo pueden acceder a los dispositivos privados las cuentas de AWS explícitamente autorizadas para hacerlo. Device Farm aísla físicamente estos dispositivos de otros dispositivos y los mantiene en una red separada.

En dispositivos administrados de forma privada, se pueden configurar las pruebas para que usen un punto de conexión de VPC para proteger las conexiones dentro y fuera de su cuenta de AWS.

Pruebas de seguridad de la infraestructura para exploradores de escritorio

Cuando utiliza la función de pruebas del explorador de escritorio, todas las sesiones de prueba se separan entre sí. Las instancias de Selenium no pueden comunicarse de forma cruzada sin un tercero intermediario, externo a AWS.

Todo el tráfico a los controladores Selenium WebDriver debe realizarse a través del extremo HTTPS generado con `createTestGridUrl`.

La característica de pruebas del explorador de escritorio no admite la configuración del punto de conexión de VPC en este momento. Usted es responsable de asegurarse de que cada instancia de prueba de Device Farm tiene acceso seguro a los recursos de la prueba.

Análisis y administración de vulnerabilidades de configuración en Device Farm

Device Farm le permite ejecutar software que el proveedor no mantenga ni repare activamente, como el proveedor del sistema operativo, el proveedor de hardware o el operador de telefonía. Device Farm hace todo lo posible por mantener el software actualizado, pero no garantiza que alguna versión concreta del software de un dispositivo físico esté actualizada, ya que su diseño permite utilizar software potencialmente vulnerable.

Por ejemplo, si se realiza una prueba en un dispositivo con Android 4.4.2, Device Farm no garantiza que el dispositivo esté parcheado contra la vulnerabilidad en Android conocida como StageFright. Depende del proveedor (y, a veces, del operador) del dispositivo proporcionar actualizaciones de seguridad a los dispositivos. No se garantiza que una aplicación maliciosa que use esta vulnerabilidad sea atrapada por nuestra cuarentena automatizada.

Los dispositivos privados se mantienen según su acuerdo con AWS.

Device Farm hace todo lo posible para evitar que las aplicaciones de los clientes realicen acciones como el rooteo o el jailbreak. Device Farm elimina los dispositivos que están en cuarentena del grupo público hasta que se hayan revisado manualmente.

Usted es responsable de mantener actualizadas todas las bibliotecas o versiones de software que utilice en sus pruebas, como Python Wheels y Ruby Gems. Device Farm recomienda actualizar las bibliotecas de prueba.

Estos recursos pueden ayudar a mantener sus dependencias de prueba actualizadas:

- Para obtener información sobre cómo proteger Ruby Gems, consulte las prácticas de seguridad en el sitio web de RubyGems.
- Para obtener información sobre el paquete de seguridad que usa Pipenv y respalda Python Packaging Authority para analizar su gráfico de dependencias en busca de vulnerabilidades conocidas, consulte Detección de vulnerabilidades de seguridad en GitHub.
- Para obtener información sobre el comprobador de dependencias Maven de Open Web Application Security Project (OWASP), consulte OWASP DependencyCheck en el sitio web de OWASP.

Es importante recordar que incluso si un sistema automatizado no cree que haya problemas de seguridad conocidos, esto no significa que no haya problemas de seguridad. Siempre use la debida diligencia cuando use bibliotecas o herramientas de terceros y verifique las firmas criptográficas cuando sea posible o razonable.

Respuesta a incidentes en Device Farm

Device Farm supervisa continuamente los dispositivos para detectar comportamientos que puedan indicar problemas de seguridad. Si AWS se informa de un caso en el que los datos de los clientes, como los resultados de las pruebas o los archivos escritos en un dispositivo público, son accesibles

por otro cliente, AWS se pone en contacto con los clientes afectados, de acuerdo con las alertas de incidentes estándar y las políticas de informes utilizadas en todos los servicios de AWS.

Registro y supervisión en Device Farm

Este servicio admite AWS CloudTrail, que es un servicio que graba sus llamadas de AWS para su Cuenta de AWS y entrega los archivos de registro a un bucket de Amazon S3. Utilice la información que recopila CloudTrail para determinar las solicitudes que se han realizado correctamente en los servicios de Servicios de AWS, quién realizó la solicitud, cuándo se realizó, etcétera. Para obtener más información sobre CloudTrail, incluido cómo activarlo y encontrar los archivos de registros, consulte la Guía del usuario de AWS CloudTrail.

Para obtener más información acerca del uso de CloudTrail con Device Farm, consulte Registro de llamadas a la API de AWS Device Farm con AWS CloudTrail.

Prácticas recomendadas de seguridad para Device Farm

Device Farm proporciona una serie de características de seguridad que debe tener en cuenta a la hora de desarrollar e implementar sus propias políticas de seguridad. Las siguientes prácticas recomendadas son directrices generales y no suponen una solución de seguridad completa. Puesto que es posible que estas prácticas recomendadas no sean adecuadas o suficientes para el entorno, considérelas como consideraciones útiles en lugar de como normas.

- Conceda a cualquier sistema de integración continua (CI) que utilice el menor privilegio posible en IAM. Plantéese el uso de credenciales temporales para cada prueba del sistema de CI para que incluso si un sistema de CI está comprometido, no pueda realizar solicitudes falsas. Para obtener más información sobre las credenciales temporales, consulte la Guía del usuario de IAM.
- Utilice comandos `adb` en un entorno de prueba personalizado para limpiar cualquier contenido que cree su aplicación. Para obtener más información sobre entornos de prueba personalizados, consulte Trabajo con entornos de pruebas personalizados.

Límites de AWS Device Farm

En la siguiente lista se describen los límites actuales de AWS Device Farm:

- El tamaño de archivo máximo de una aplicación que puede cargar es de 4 GB.
- No existe ningún límite al número de dispositivos que se puede incluir en la ejecución de una prueba. Sin embargo, el número máximo de dispositivos que Device Farm probará simultáneamente durante una prueba de funcionamiento es cinco. (Este número se puede aumentar bajo demanda).
- No existe ningún límite en el número de ejecuciones que puede programar.
- Hay un límite de 150 minutos para la duración de una sesión de acceso remoto.
- Hay un límite de 150 minutos para la duración de una prueba de funcionamiento automatizada.
- El número máximo de trabajos en curso, incluidos los trabajos pendientes en cola en su cuenta, es de 250. Este es un límite variable.
- No existe ningún límite al número de dispositivos que se puede incluir en la ejecución de una prueba. La cantidad de dispositivos o trabajos en los que se pueden ejecutar pruebas en paralelo en un momento determinado es igual a la simultaneidad a nivel de cuenta. La simultaneidad predeterminada a nivel de cuenta para el uso medido en AWS Device Farm es 5. Puede solicitar un aumento de este número hasta un umbral determinado en función del caso de uso. La simultaneidad predeterminada a nivel de cuenta para un uso no medido es igual al número de ranuras a los que se está suscrito para esa plataforma.

Herramientas y complementos de AWS Device Farm

En esta sección se incluyen enlaces e información acerca de cómo trabajar con herramientas y complementos de AWS Device Farm. Puede encontrar complementos de Device Farm en Laboratorios de AWS en GitHub.

Si es un desarrollador de Android, también disponemos de una aplicación de ejemplo de AWS Device Farm para Android en GitHub. Puede utilizar la aplicación y las pruebas de ejemplo como referencia para sus propios scripts de pruebas de Device Farm.

Temas

Integración de AWS Device Farm con el complemento Jenkins CI

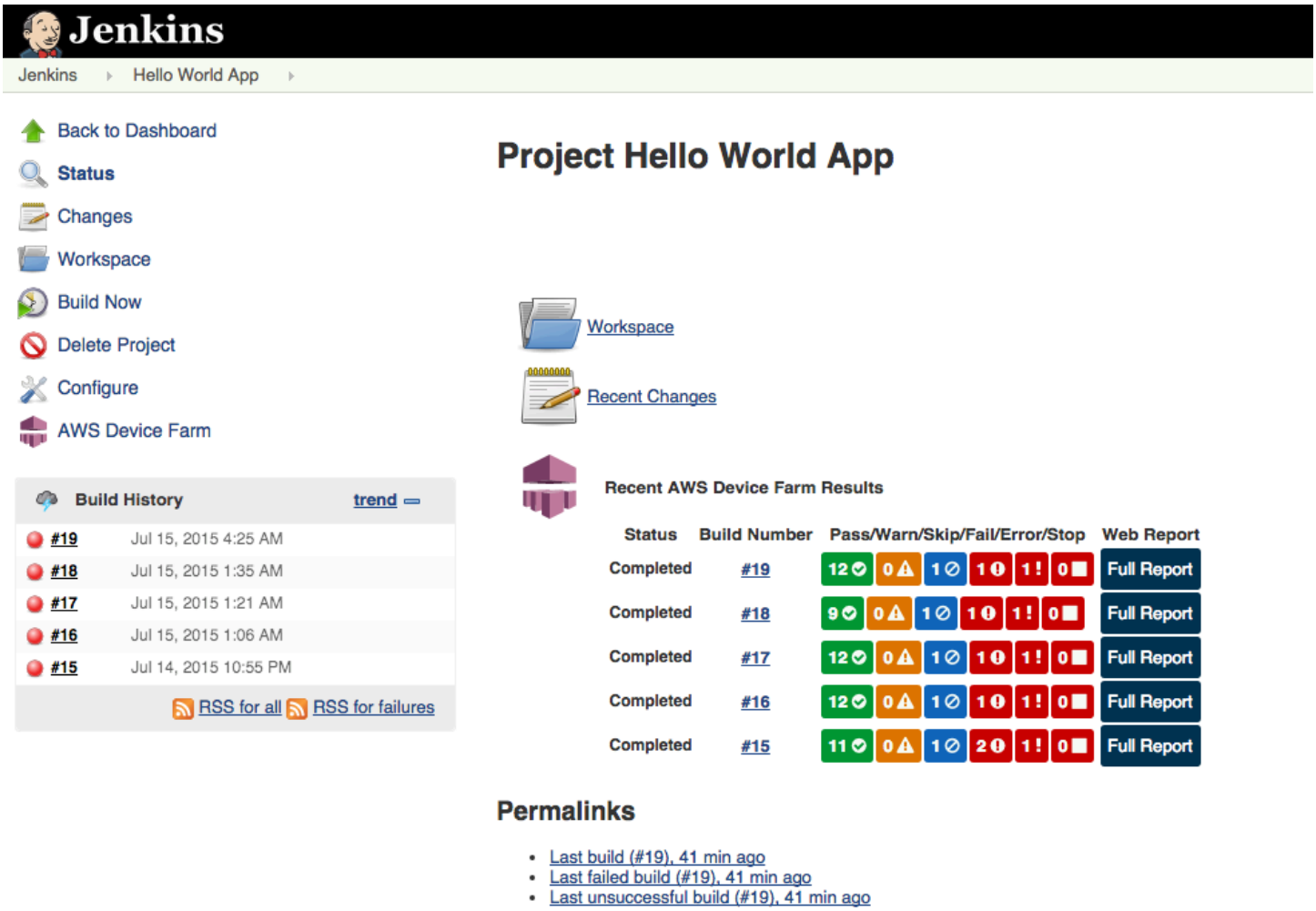
Este complemento proporciona funcionalidad de AWS Device Farm desde su propio servidor Jenkins de integración continua (CI). Para obtener más información, consulte Jenkins (software).

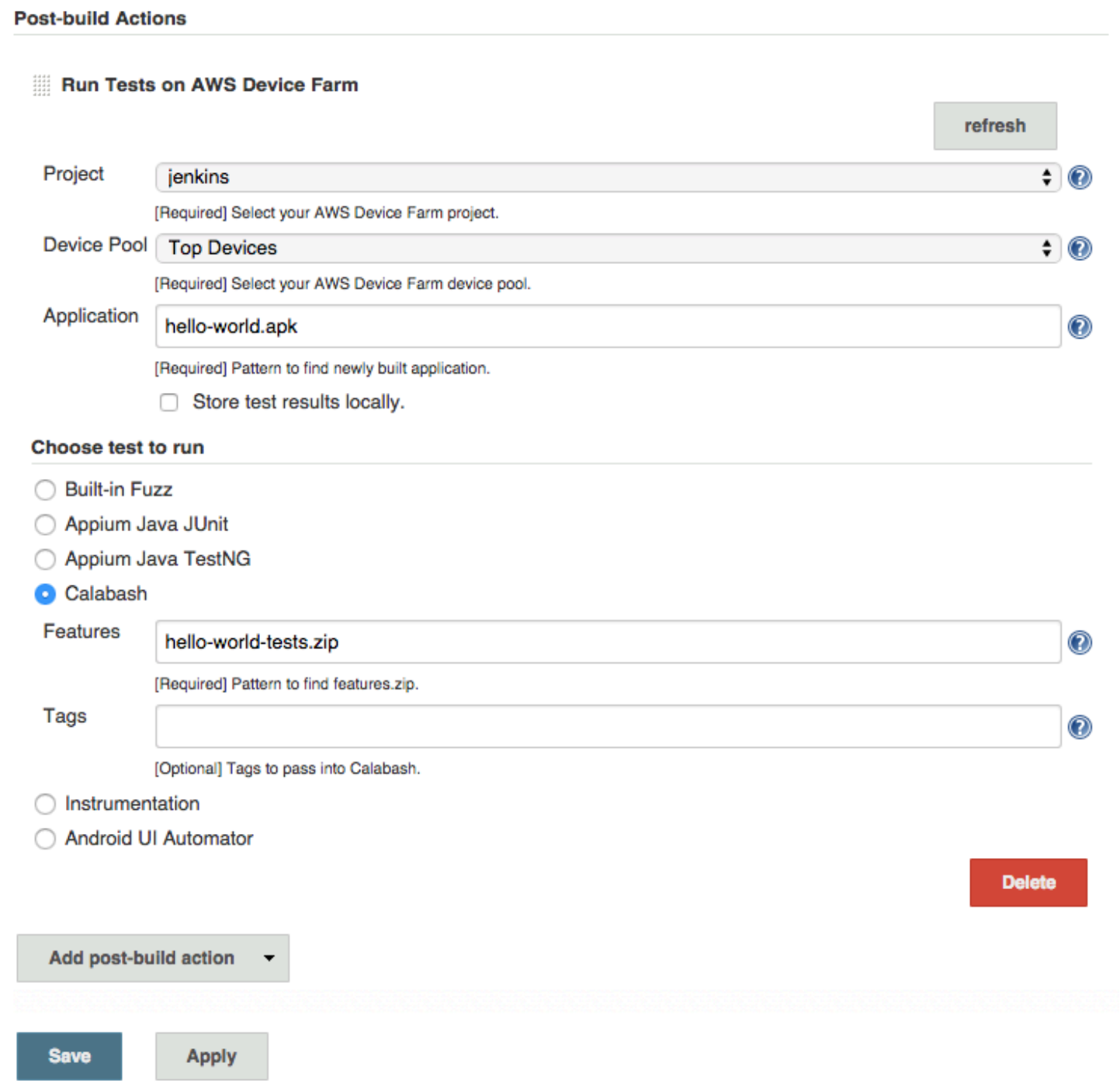
> **Note**
>
> Para descargar el complemento Jenkins, vaya a GitHub y siga las instrucciones de Paso 1: Instalación del complemento.

Esta sección contiene una serie de procedimientos para configurar y utilizar el complemento Jenkins CI con AWS Device Farm.

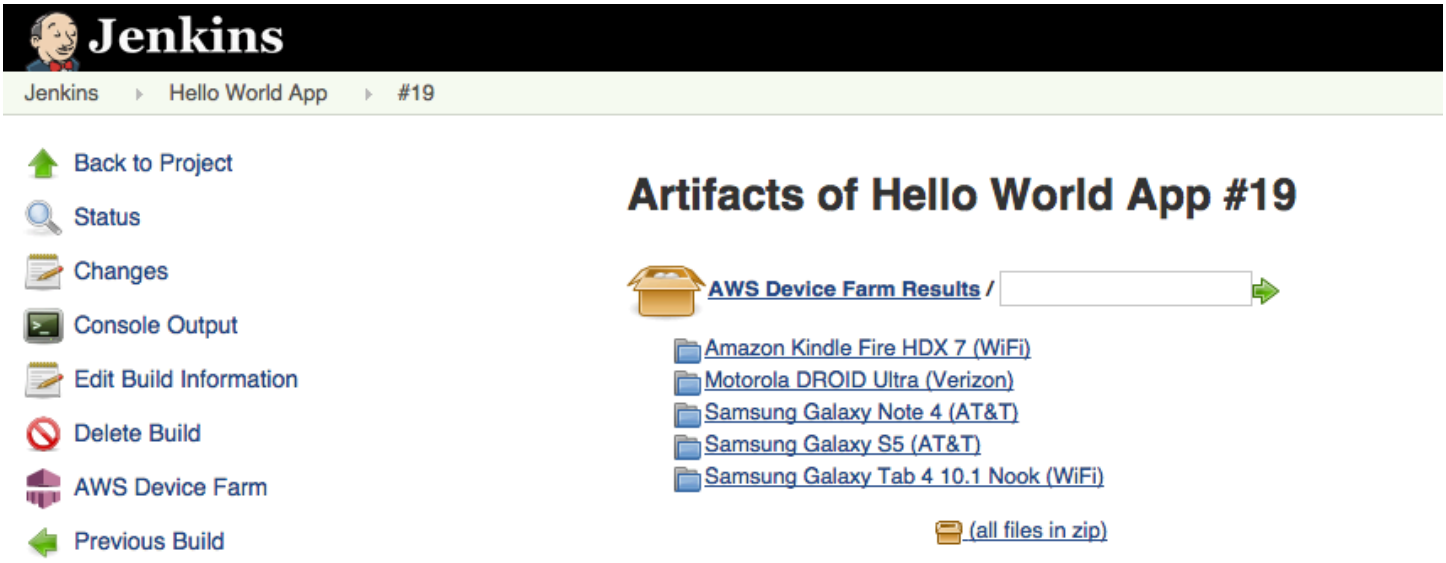
Temas

Las siguientes imágenes muestran las características del complemento Jenkins CI.





El complemento también puede extraer todos los artefactos de las pruebas (registros, capturas de pantalla, etc.) localmente:



Paso 1: Instalación del complemento

Existen dos opciones para instalar el complemento de integración continua (CI) de Jenkins para AWS Device Farm. Puede buscar el complemento desde el cuadro de diálogo Complementos disponibles en la interfaz de usuario web de Jenkins, o bien puede descargar el archivo `hpi` e instalarlo desde Jenkins.

Instalación desde la interfaz de usuario de Jenkins

1. Para encontrar el complemento en la interfaz de usuario de Jenkins, seleccione Administrar Jenkins, Administrar dispositivos y, a continuación, seleccione Disponibles.
2. Busque aws-device-farm.
3. Instale el complemento AWS Device Farm.
4. Asegúrese de que el complemento es propiedad del usuario de `Jenkins`.
5. Reinicie Jenkins.

Descarga del complemento

1. Descargue el archivo `hpi` directamente desde http://updates.jenkins-ci.org/latest/aws-device-farm.hpi.
2. Asegúrese de que el complemento es propiedad del usuario de `Jenkins`.
3. Instale el complemento mediante una de las siguientes opciones:

- Para cargar el complemento, seleccione Administrar Jenkins, Administrar complementos, Avanzados y, a continuación, seleccione Cargar complemento.
- Coloque el archivo `hpi` en el directorio del complemento de Jenkins (normalmente `/var/lib/jenkins/plugins`).

4. Reinicie Jenkins.

Paso 2: Crear un usuario de AWS Identity and Access Management para el complemento Jenkins CI

Le recomendamos que no use su cuenta raíz de AWS para obtener acceso a Device Farm. En su lugar, cree un usuario de AWS Identity and Access Management (IAM) (o utilice un usuario de IAM existente) en su cuenta de AWS y, a continuación, obtenga acceso a Device Farm con ese usuario de IAM.

Para crear un nuevo usuario de IAM, consulte Crear un usuario de IAM (AWS Management Console). Asegúrese de generar una clave de acceso para cada usuario y de descargar o guardar las credenciales de seguridad de los usuarios. Necesitará estas credenciales más tarde.

Conceda permiso al usuario de IAM para obtener acceso a Device Farm.

Para conceder permiso al usuario de IAM para obtener acceso a Device Farm, cree una nueva política de acceso en IAM. Después, asigne la política de acceso al usuario de IAM como se indica a continuación.

> **Note**
>
> La cuenta raíz de AWS o el usuario de IAM que utilice para completar los pasos siguientes deben tener permiso para crear la siguiente política de IAM y asociársela al usuario de IAM. Para obtener más información, consulte Administración de políticas de IAM

Para crear la política de acceso en IAM

1. Abra la consola de IAM en https://console.aws.amazon.com/iam/.
2. Seleccione Políticas.
3. Seleccione Crear política. Si aparece el botón Empezar, selecciónelo y, a continuación, seleccione Crear política.

4. Junto a Create Your Own Policy, seleccione Select.
5. En Nombre de política, escriba un nombre para la política (por ejemplo, **`AWSDeviceFarmAccessPolicy`**).
6. Para Descripción, escriba una descripción que le ayude a asociar este usuario de IAM con el proyecto Jenkins.
7. En Documento de política, escriba la siguiente instrucción:

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

8. Seleccione Crear política.

Para asignar la política de acceso al usuario de IAM

1. Abra la consola de IAM en https://console.aws.amazon.com/iam/.
2. Seleccione Usuarios.
3. Seleccione el usuario de IAM a quien desea asignar la política de acceso.
4. En el área Permisos, en Políticas administradas, seleccione Asociar política.
5. Seleccione la política que acaba de crear (por ejemplo, AWSDeviceFarmAccessPolicy).
6. Seleccione Asociar política.

Paso 3: Instrucciones para realizar la configuración por primera vez

La primera vez que ejecute el servidor de Jenkins, tendrá que configurar el sistema como se indica a continuación.

> **Note**
>
> Si utiliza ranuras de dispositivos, la característica de ranuras de dispositivos está deshabilitada de forma predeterminada.

1. Inicie sesión en la interfaz de usuario web de Jenkins.
2. En la parte izquierda de la pantalla, seleccione Administrar Jenkins.
3. Seleccione Configurar sistema.
4. Desplácese hacia abajo hasta el encabezado AWS Device Farm.
5. Copie sus credenciales de seguridad Paso 2: Crear un usuario de IAM y pegue el ID de clave de acceso y la clave de acceso secreta en sus respectivas casillas.
6. Seleccione Save.

Paso 4: Usar el complemento en un trabajo de Jenkins

Una vez que haya instalado el complemento Jenkins, siga estas instrucciones para utilizar el complemento en un trabajo de Jenkins.

1. Inicie sesión en la interfaz de usuario web de Jenkins.
2. Haga clic en el trabajo que desea editar.
3. En la parte izquierda de la pantalla, seleccione Configurar.
4. Desplácese hacia abajo hasta el encabezado Acciones posteriores a la compilación.
5. Haga clic en Añadir acción posterior a la compilación y seleccione Ejecutar pruebas en AWS Device Farm.
6. Seleccione el proyecto que desearía usar.
7. Seleccione el grupo de dispositivos que desearía usar.
8. Seleccione si desea que los elementos de las pruebas (como los registros y las capturas de pantalla) se archiven localmente.
9. En Aplicación, rellene la ruta de la aplicación compilada.
10. Seleccione la prueba que desea ejecutar y rellene todos los campos obligatorios.
11. Seleccione Save.

Dependencias

El complemento Jenkins CI requiere el SDK para móviles de AWS 1.10.5 o posterior. Para obtener más información y para instalar el SDK, consulte AWS Mobile SDK.

Complemento Gradle de AWS Device Farm

Este complemento proporciona la integración de AWS Device Farm con el sistema de compilación Gradle en Android Studio. Para obtener más información, consulte Gradle.

> **Note**
>
> Para descargar el complemento Gradle, vaya a GitHub y siga las instrucciones de Creación del complemento Gradle de Device Farm.

El complemento Gradle de Device Farm proporciona funcionalidad de Device Farm desde su entorno de Android Studio. Puede iniciar pruebas en teléfonos y tabletas Android reales alojados en Device Farm.

Esta sección contiene una serie de procedimientos para configurar y utilizar el complemento Gradle de Device Farm.

Temas

Paso 1: Crear el complemento Gradle de AWS Device Farm

Este complemento proporciona la integración de AWS Device Farm con el sistema de compilación Gradle en Android Studio. Para obtener más información, consulte Gradle.

> **Note**
>
> La creación del complemento es opcional. El complemento se publica a través de Maven Central. Si desea permitir que Gradle descargue el complemento directamente, omita este paso y vaya a Paso 2: Configurar el complemento Gradle de AWS Device Farm.

Para crear el complemento

1. Vaya a GitHub y clone el repositorio.
2. Cree el complemento mediante `gradle install`.

   El complemento se instala en el repositorio de Maven local.

Paso siguiente: Paso 2: Configurar el complemento Gradle de AWS Device Farm

Paso 2: Configurar el complemento Gradle de AWS Device Farm

Si aún no lo ha hecho, clone el repositorio e instale el complemento mediante el siguiente procedimiento: Creación del complemento Gradle de Device Farm.

Para configurar el complemento AWS Device Farm para Gradle

1. Añada el elemento del complemento a la lista de dependencia en `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 el complemento en el archivo `build.gradle`. La siguiente configuración específica de pruebas debería servirle como guía:

```
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. Ejecute su prueba de Device Farm mediante la siguiente tarea: `gradle devicefarmUpload`.

   La salida de compilación mostrará un enlace en la consola de Device Farm donde puede monitorizar la ejecución de las pruebas.

Paso siguiente: Generar un usuario de IAM

Paso 3: Generar un usuario de IAM

AWS Identity and Access Management (IAM) le ayuda a administrar los permisos y las políticas para trabajar con recursos de AWS. Este tema le muestra el proceso para generar un usuario de IAM con permisos para obtener acceso a los recursos de AWS Device Farm.

Si aún no lo ha hecho, complete los pasos 1 y 2 antes de generar un usuario de IAM.

Le recomendamos que no use su cuenta raíz de AWS para obtener acceso a Device Farm. En su lugar, cree un usuario de IAM (o utilice un usuario de IAM existente) en su cuenta de AWS y, a continuación, obtenga acceso a Device Farm con ese usuario de IAM.

> **Note**
>
> La cuenta raíz de AWS o el usuario de IAM que utilice para completar los pasos siguientes deben tener permiso para crear la siguiente política de IAM y asociársela al usuario de IAM. Para obtener más información, consulte Administración de políticas de IAM.

Para crear un nuevo usuario con la política de acceso adecuada en IAM

1. Abra la consola de IAM en https://console.aws.amazon.com/iam/.
2. Seleccione Usuarios.
3. Seleccione Crear nuevos usuarios.
4. Introduzca el nombre de usuario de su elección.

   Por ejemplo, **`GradleUser`**.
5. Seleccione Crear.
6. Seleccione Descargar credenciales y guárdelas en una ubicación donde pueda recuperarlas fácilmente más adelante.
7. Elija Cerrar.
8. Elija el nombre de usuario en la lista.
9. En Permisos, expanda el encabezado Políticas insertadas haciendo clic en la flecha hacia abajo situada a la derecha.
10. Seleccione Haga clic aquí donde dice No hay políticas insertadas que mostrar. Para crear una, haga clic aquí.

11. En la pantalla Establecer permisos, seleccione Política personalizada.
12. Elija Select.
13. Especifique un nombre para la política, como **`AWSDeviceFarmGradlePolicy`**.
14. En Documento de política, pegue la siguiente política.

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

15. Seleccione Apply Policy.

Paso siguiente: Configurar los tipos de prueba.

Para obtener más información, consulte Creación de usuarios de IAM AWS Management Console () o Configuración.

Paso 4: Configurar los tipos de prueba

De forma predeterminada, el complemento Gradle de AWS Device Farm ejecuta la prueba Trabajar con instrumentación para Android y AWS Device Farm. Si desea ejecutar sus propias pruebas o especificar parámetros adicionales, puede elegir configurar un tipo de la prueba. En este tema se ofrece información sobre cada tipo de prueba disponible y qué se debe hacer en Android Studio con el fin de configurarlo para su uso. Para obtener más información acerca de los tipos de pruebas disponibles en Device Farm, consulte Trabajar con tipos de pruebas en AWS Device Farm.

Si aún no lo ha hecho, complete los pasos 1 y 3 antes de configurar los tipos de pruebas.

> **Note**
>
> Si utiliza ranuras de dispositivos, la característica de ranuras de dispositivos está deshabilitada de forma predeterminada.

Appium

Device Farm ofrece soporte para Appium Java JUnit y TestNG para Android.

- Appium (en Java (JUnit))
- Appium (en Java (TestNG))

Puede elegir `useTestNG()` o `useJUnit()`. `JUnit` es el valor predeterminado y no es preciso especificarlo de forma explícita.

```
    appium {
        tests file("path to zip file") // required
        useTestNG() // or useJUnit()
    }
```

Integrado: fuzzing

Device Farm proporciona un tipo de prueba de difusión integrada, que envía de forma aleatoria eventos de interfaz de usuario a los dispositivos y, a continuación, crea un informe con los resultados.

```
    fuzz {

       eventThrottle 50 // optional default
       eventCount 6000  // optional default
       randomizerSeed 1234 // optional default blank

     }
```

Para obtener más información, consulte Integrado: fuzzing (Android e iOS).

Instrumentación

Device Farm ofrece soporte para instrumentación (JUnit, Espresso, Robotium o cualquier otra prueba basada en instrumentación) para Android. Para obtener más información, consulte Trabajar con instrumentación para Android y AWS Device Farm.

Cuando se ejecuta una prueba de instrumentación en Gradle, Device Farm utiliza el archivo `.apk` generado desde el directorio androidTest como el fuente de las pruebas.

```
    instrumentation {

        filter "test filter per developer docs" // optional

    }
```

Dependencias

Tiempo de ejecución

- El complemento Gradle de Devuce Farn requiere AWS Mobile SDK 1.10.15 o posterior. Para obtener más información y para instalar el SDK, consulte AWS Mobile SDK.
- Android tools builder test api 0.5.2
- Apache Commons Lang3 3.3.4

Para pruebas unitarias

- Testng 6.8.8
- Jmockit 1.19
- Android gradle tools 1.3.0

Historial del documento

En la siguiente tabla se describen los cambios importantes que se han realizado en la documentación desde la última versión esta guía.

Cambio	Descripción	Fecha de modificación
Compatibilidad de AL2	Ahora, Device Farm es compatible con el entorno de pruebas AL2 para Android. Más información sobre AL2.	6 de noviembre de 2023
Migración de entornos de prueba estándar a entornos de prueba personalizados	Se ha actualizado la guía de migración para documentar la obsolescencia de las pruebas de modo estándar en diciembre de 2023.	3 de septiembre de 2023
Compatibilidad de VPC ENI	Ahora, Device Farm permite a los dispositivos privados utilizar la característica de conectividad VPC-ENI para ayudar a los clientes a conectarse de forma segura a sus puntos de conexión privados con host en AWS, un software en las instalaciones, o a otro proveedor en la nube. Más información sobre VPC-ENI.	15 de mayo de 2023
Actualizaciones de Polaris UI	La consola Device Farm ahora es compatible con la estructura Polaris.	28 de julio de 2021
Compatibilidad con Python 3	Device Farm ahora es compatible con Python 3 en pruebas de modo personalizado. Obtenga más información sobre el uso de Python 3 en sus paquetes de prueba: • Appium (Python) • Appium (Python)	20 de abril de 2020
Nueva información de seguridad e información sobre	Para facilitar y ampliar la seguridad de los servicios de AWS, se ha creado una nueva sección sobre seguridad	27 de marzo de 2020

Cambio	Descripción	Fecha de modificación
los recursos de etiquetado de AWS.	. Para obtener más información, consulte Seguridad en AWS Device Farm Se ha añadido una nueva sección sobre etiquetado en Device Farm. Para obtener más información acerca del etiquetado, consulte Etiquetado en Device Farm.	
Eliminación del acceso directo al dispositivo.	El acceso directo a dispositivos (depuración remota en dispositivos privados) ya no está disponible para uso general. Si tiene dudas sobre la disponibilidad del acceso directo a dispositivos en el futuro, contacte con nosotros.	9 de septiembre de 2019
Configuración del complemento Gradle actualizada	La configuración revisada del complemento Gradle incluye ahora una versión personalizable de la configuración de gradle, con comentarios sobre los parámetros opcionales. Obtener más información sobre Configuración del complemento Device Farm para Gradle.	16 de agosto de 2019
Nuevo requisito de ejecuciones de prueba con XCTest	Para las ejecuciones de prueba que utilizan el marco XCTest, Device Farm ahora requiere un paquete de la aplicación que se haya compilado para pruebas. Obtener más información sobre the section called “XCTest”.	4 de febrero de 2019
Compatibilidad con tipos de prueba de Appium Node.js y Appium Ruby en entornos personalizados	Ahora puede ejecutar sus pruebas en los entornos de prueba personalizados de Appium Node.js y Appium Ruby. Obtener más información sobre Trabajar con tipos de pruebas en AWS Device Farm.	10 de enero de 2019

Cambio	Descripción	Fecha de modificación
Compatibilidad con Appium server versión 1.7.2 en entornos estándar y personalizados. Compatibilidad con la versión 1.8.1 si se usa un archivo YAML de especificación de prueba personalizada en un entorno de pruebas personalizado.	Ahora, puede ejecutar las pruebas en entornos de pruebas estándar y personalizados con Appium server versiones 1.72, 1.71 y 1.6.5. También puede ejecutar las pruebas con las versiones 1.8.1 y 1.8.0 mediante un archivo YAML de especificación de prueba personalizada en un entorno de pruebas personalizado. Obtener más información sobre Trabajar con tipos de pruebas en AWS Device Farm.	2 de octubre de 2018
Entornos de pruebas personalizados	Con un entorno de pruebas personalizado, puede asegurarse de que sus pruebas se ejecuten como en su entorno local. Device Farm ahora admite registros en directo y transmisión de vídeo, de modo que puede obtener comentarios instantáneos sobre las pruebas que se ejecutan en un entorno de pruebas personalizado. Obtener más información sobre Trabajo con entornos de pruebas personalizados.	16 de agosto de 2018
Compatibilidad con el uso de Device Farm como proveedor de pruebas de AWS CodePipeline.	Ahora puede configurar una canalización AWS CodePipeline para utilizar las ejecuciones de AWS Device Farm como acciones de prueba en su proceso de lanzamiento. CodePipeline le permite vincular rápidamente su repositorio a las etapas de creación y prueba para lograr un sistema de integración continua personalizado según sus necesidades. Obtener más información sobre Uso de AWS Device Farm en una etapa de prueba de CodePipeline.	19 de julio de 2018

Cambio	Descripción	Fecha de modificación
Soporte para dispositivos privados	Ahora puede utilizar dispositivos privados para programar ejecuciones de prueba e iniciar sesiones de acceso remoto. Puede administrar los perfiles y las configura ciones de estos dispositivos, crear puntos de conexión de VPC de mazon para probar aplicaciones privadas y crear las sesiones de depuración remota. Obtener más información sobre Trabajar con dispositivos privados en AWS Device Farm.	2 de mayo de 2018
Soporte para Appium 1.6.3	Ahora ya puede establecer la versión de Appium para sus pruebas personalizadas de Appium.	21 de marzo de 2017
Establecimiento del tiempo de espera de ejecución para ejecuciones de prueba	Puede establecer el tiempo de espera de ejecución para una ejecución de prueba o para todas las pruebas en un proyecto. Obtener más información sobre Establecimiento del tiempo de espera de ejecución para ejecuciones de pruebas en AWS Device Farm.	9 de febrero de 2017
Forma de red	Puede simular conexiones y condiciones de una de red para una ejecución de prueba. Obtener más información sobre Simulación de conexiones y condiciones de una de red para ejecuciones de AWS Device Farm.	8 de diciembre de 2016
Nueva sección de solución de problemas	Ahora puede solucionar problemas de cargas de paquetes de pruebas mediante un conjunto de procedimientos diseñado para resolver mensajes de error que pudiera encontrar en la consola de Device Farm. Obtener más información sobre Solución de problemas de Device Farm.	10 de agosto de 2016
Sesiones de acceso remoto	Ahora puede obtener acceso de forma remota e interactu ar con un único dispositivo en la consola. Obtener más información sobre Trabajar con acceso remoto.	19 de abril de 2016

Cambio	Descripción	Fecha de modificación
Autoservicio de ranuras de dispositivos	Puede adquirir ranuras de dispositivos con la AWS Management Console, la AWS Command Line Interface o la API. Obtenga más información sobre cómo Adquisición de una ranura para dispositivos en Device Farm.	22 de marzo de 2016
Cómo detener ejecuciones de prueba	Ahora puede parar ejecuciones de prueba con la AWS Management Console, la AWS Command Line Interface o la API. Obtenga más información sobre cómo Detener una ejecución en AWS Device Farm.	22 de marzo de 2016
Nuevos tipos de pruebas de XCTest UI	Ahora puede ejecutar pruebas personalizadas de XCTest UI en aplicaciones iOS. Obtenga más información sobre el tipo de prueba de XCTest UI.	8 de marzo de 2016
Nuevos tipos de pruebas de Appium Python	Ahora puede ejecutar pruebas personalizadas de Appium Python en aplicaciones Android, iOS y web. Obtener más información sobre Trabajar con tipos de pruebas en AWS Device Farm.	19 de enero de 2016
Tipos de pruebas para aplicaciones web	Ahora puede ejecutar pruebas personalizadas de Appium Java JUnit y TestNG en aplicaciones web. Obtener más información sobre Trabajar con pruebas de aplicaciones web en AWS Device Farm.	19 de noviembre de 2015
Complemento de Gradle para AWS Device Farm	Obtenga más información sobre cómo instalar y utilizar el Complemento Gradle de Device Farm.	28 de septiembre de 2015
Nueva prueba integrada para Android: Explorer	La prueba de explorador rastrea la aplicación analizando todas las pantallas como si fuera un usuario final, y toma capturas de pantalla a medida que la explora.	16 de septiembre de 2015
Se ha añadido compatibilidad con iOS	Obtenga más información acerca de probar dispositivos iOS y ejecutar pruebas de iOS (incluidas XCTest) en Trabajar con tipos de pruebas en AWS Device Farm.	4 de agosto de 2015

Cambio	Descripción	Fecha de modificación
Versión pública inicial	Esta es la versión pública inicial de la Guía para desarrolladores de AWS Device Farm.	13 de julio de 2015

Glosario de AWS

Para ver la terminología más reciente de AWS, consulte el Glosario de AWS en la Referencia de Glosario de AWS.

Las traducciones son generadas a través de traducción automática. En caso de conflicto entre la traducción y la version original de inglés, prevalecerá la version en inglés.