aws

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:

- Acceda de forma remota a un dispositivo desde su ordenador local, ya sea de forma interactiva en su navegador web o pruébelo automáticamente con Appium desde un cliente local.
- Ejecuta automáticamente las pruebas de las aplicaciones mediante el entorno de ejecución de pruebas gestionado de Device Farm.

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

Acceso remoto

El acceso remoto le permite interactuar con un dispositivo a través de su navegador web en tiempo real. El acceso remoto también le permite ejecutar pruebas de Appium desde su cliente local en dispositivos Device Farm remotos mediante un punto final de Appium gestionado.

La interacción en tiempo real con un dispositivo puede resultar útil en varios escenarios, como probar aplicaciones manualmente, reproducir errores en un dispositivo específico, comprobar la representación visual de la aplicación en diferentes tipos de pantallas y secuencias de instalación y actualización de la aplicación. El terminal de Appium de Device Farm, totalmente gestionado, te permite desarrollar, probar y depurar tus pruebas de Appium, además de proporcionarte una respuesta rápida.

El terminal Appium admite cualquier idioma de su elección, cualquier IDE local, depuración en directo con puntos de interrupción, vídeo y registros en directo y herramientas como Appium Inspector. Puedes ejecutar pruebas tantas veces como desees en el mismo dispositivo durante tu sesión de acceso remoto con un límite de 150 minutos.

Durante una sesión de acceso remoto, Device Farm registra los detalles sobre las acciones que se llevan a cabo al interactuar con el dispositivo. Al final de la sesión, se generan registros con estos detalles y una captura de video de la sesión.

Pruebas de aplicaciones automatizadas

Device Farm te permite ejecutar pruebas automatizadas en varios dispositivos en paralelo al cargar la aplicación y las pruebas. Las pruebas se ejecutan automáticamente en un entorno totalmente gestionado en los hosts de prueba que se pueden configurar como un archivo de especificaciones de prueba. El entorno utiliza los hosts de prueba de Device Farm, por lo que no tiene que preocuparse por aprovisionar su propia infraestructura para ejecutar las pruebas. Los hosts y dispositivos de prueba se pueden conectar de forma segura a su VPC para acceder a sus puntos finales privados.

A medida que se completan las pruebas, se genera un informe de prueba que contiene los resultados de alto nivel, los registros de bajo nivel, las capturas de pantalla y los artefactos de la prueba.

Device Farm permite probar aplicaciones nativas e híbridas para Android e iOS. Para obtener más información acerca de los tipos de pruebas admitidos, consulte Marcos de pruebas y pruebas integradas en AWS Device Farm.

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.

job

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.

proyecto

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.

sesión

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.

prueba

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 suscribirse a una Cuenta de AWS

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

   Parte del procedimiento de registro consiste en recibir una llamada telefónica o mensaje de texto 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 y utilice únicamente el usuario raíz para realizar 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 (Consola de administración de AWS).

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*

   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. Necesitas una AWS cuenta y un usuario AWS Identity and Access Management (IAM) con permiso para acceder a Device Farm.

- Para Android, puede traer un archivo .apk (paquete de aplicaciones Android) o usar la aplicación de muestra que proporcionamos. 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 Marcos de pruebas y pruebas integradas 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 Crear 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 pestaña Pruebas automatizadas, seleccione Crear ejecución. De forma alternativa, puede seguir el tutorial de la consola y seleccionar Crear ejecución con el tutorial.
2. (Opcional) En Configuración de ejecución, en la sección Nombre de ejecución, escriba un nombre para la ejecución. Si no se proporciona ningún nombre, la consola de Device Farm asignará a la ejecución el nombre “My Device Farm run” de forma predeterminada.

3. En Configuración de ejecución, en la sección Tipo de ejecución, seleccione el tipo de ejecución. Seleccione Aplicación Android si no tiene ninguna aplicación lista para probarla o si está probando una aplicación en Android (.apk). Seleccione Aplicación iOS si está probando una aplicación iOS (.ipa).
4. En Seleccionar aplicación, en la sección Opciones de selección de aplicaciones, elija Seleccionar aplicación de muestra proporcionada por Device Farm si no tiene ninguna aplicación disponible para probarla. Si va a traer su propia aplicación, seleccione Cargar aplicación propia y elija el archivo de su aplicación. Si carga una aplicación iOS, asegúrese de elegir Dispositivo iOS, en lugar de un simulador.
5. En la página Configurar prueba, en Seleccionar marco de pruebas, elija uno de los marcos de prueba o conjuntos de pruebas integrados. Para obtener más información acerca de cada opción, consulte Marcos de pruebas y pruebas integradas.
   - 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. En Seleccionar dispositivos, seleccione Usar grupo de dispositivos y Dispositivos de escritorio.
7. (Opcional) Para añadir una configuración adicional, abra el menú desplegable Configuración adicional. En esta sección, puede hacer una de las siguientes acciones:
   - 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.
- Seleccione Habilitar grabación de video para grabar video durante la prueba.
- Seleccione Habilitar la captura de datos de rendimiento de aplicaciones para capturar datos de desempeño en el dispositivo.

> **Note**
>
> Por el momento, la configuración del estado de radio y la configuración regional del dispositivo solo está disponible en pruebas nativas de Android.

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

8. En la parte inferior de la página, elija Crear ejecución para programar la 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 Visualización de informes de pruebas en Device Farm.

Siguientes pasos

Para obtener más información sobre Device Farm, consulte Conceptos.

Adquisición de una ranura de dispositivos en Device Farm

Puedes usar la consola Device Farm, AWS Command Line Interface (AWS CLI) o la API Device Farm para comprar una ranura para dispositivos.

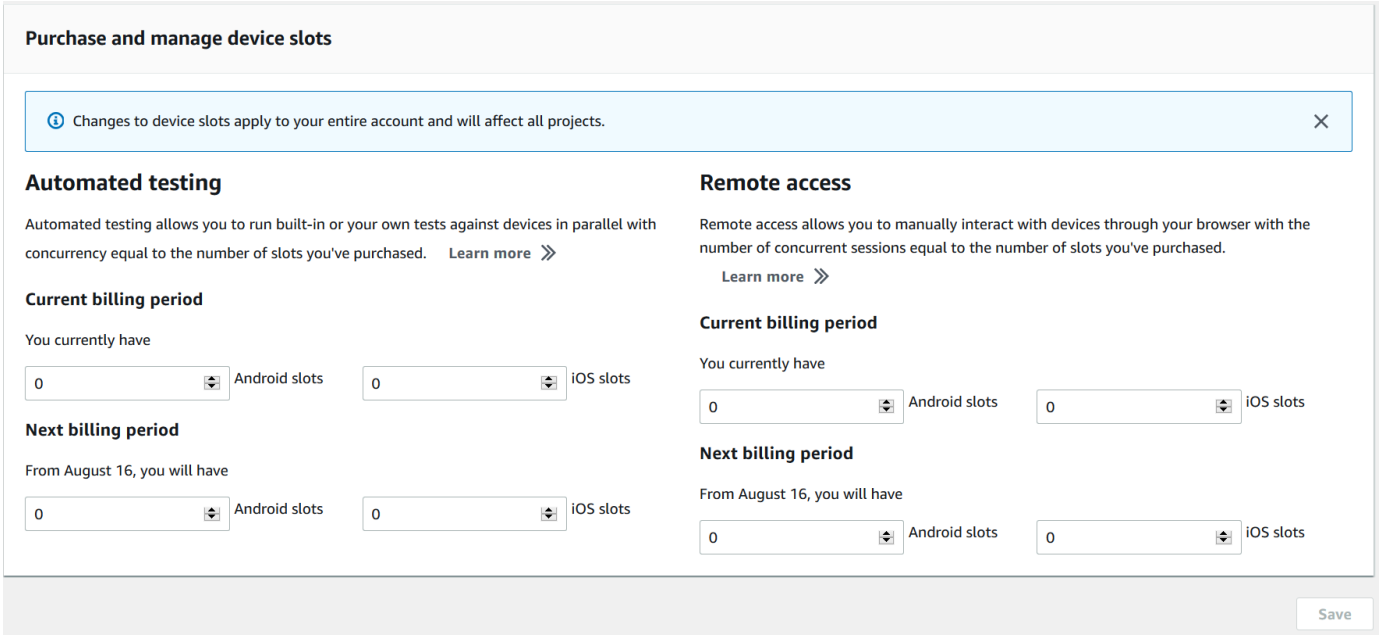
Adquisición de ranuras de dispositivos (consola)

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.

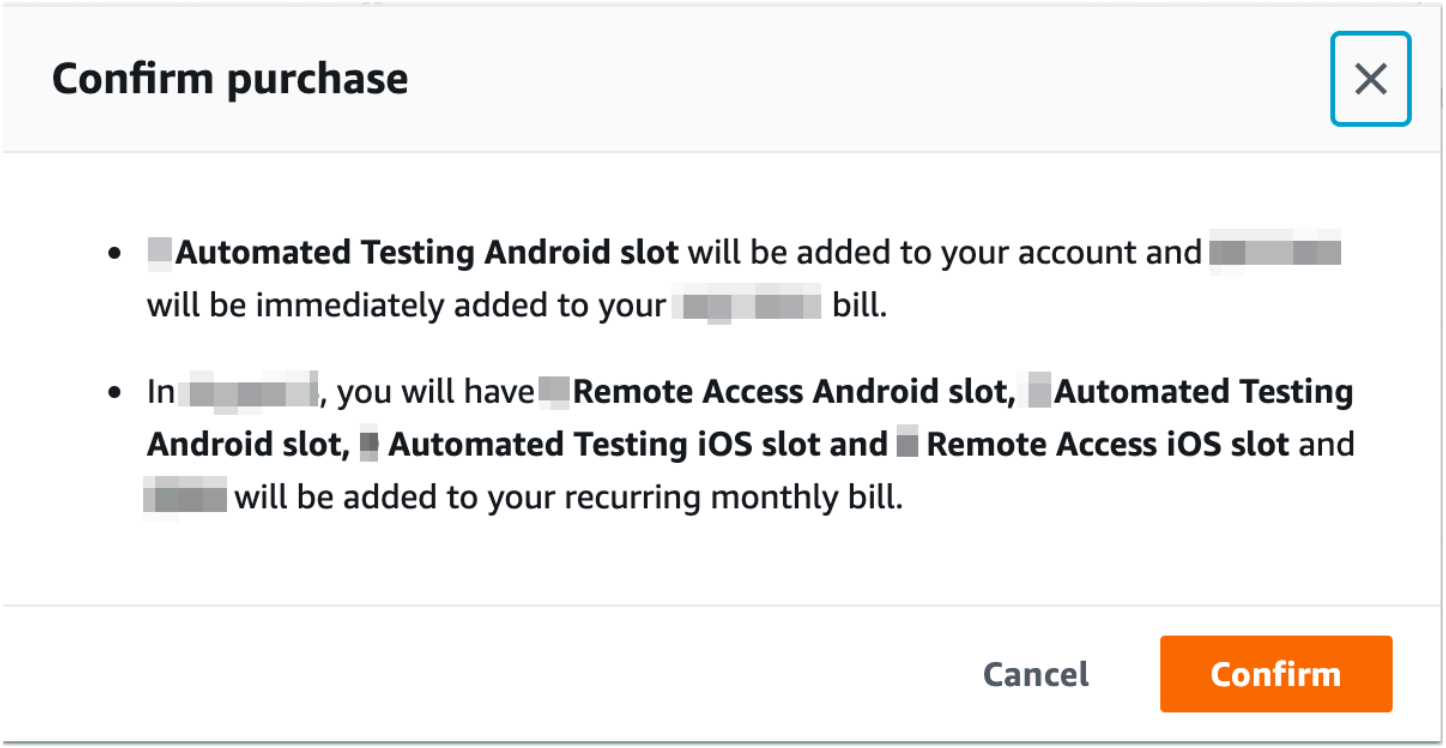
   A medida que cambie el importe de las ranuras, el texto se actualiza dinámicamente con el importe de facturación. Para obtener más información, consulte Precios de AWS Device Farm.

   > **Important**
   >
   > Si cambias el número de ranuras para dispositivos pero ves un mensaje de contacto o contacto con nosotros para realizar una compra, significa que tu AWS cuenta aún no está autorizada a comprar el número de ranuras para dispositivos que solicitaste. 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. Aparecerá 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 la AWS CLIReferencia de .

Adquisición de una ranura de dispositivos (API)

1. Llama a la GetAccountSettingsoperadora para ver la configuración de tu cuenta.
2. Llame a la ListOfferingsoperación para que le indiquen las ofertas de ranuras para dispositivos que tiene disponibles.
3. Llame a la ListOfferingPromotionsoperación para obtener una lista de las ofertas y promociones disponibles.

   > **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. Llame a la PurchaseOfferingoperación para comprar una oferta.
5. Llame a la GetOfferingStatusoperación para obtener el estado de la oferta.

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.

Cancelación de una ranura de dispositivos en Device Farm

Puede cancelar la cantidad de ranuras de dispositivos tanto para las pruebas automatizadas como para el acceso remoto. Para obtener instrucciones, consulte una de las siguientes secciones. El importe cargado a su cuenta para el siguiente ciclo de facturación aparecerá debajo del campo del período de facturación.

Para obtener más información acerca de las ranuras de dispositivos, consulte Adquisición de una ranura de dispositivos en Device Farm.

Cancelación de una ranura 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 reducir el número de ranuras para dispositivos, tanto para las pruebas automatizadas como para el acceso remoto, rebajando el valor correspondiente en Próximo período de facturación. El importe cargado a su cuenta para el siguiente ciclo de facturación aparecerá debajo del campo del período de facturación.
4. Seleccione Save. Aparecerá la ventana Confirmar cambio. Revise la información y, a continuación, seleccione Confirmar para completar la transacción.

Cancelar una ranura de dispositivo (AWS CLI)

Puede ejecutar el comando renew-offering para cambiar la cantidad de dispositivos para el siguiente ciclo de facturación.

Cancelación de una ranura de dispositivos (API)

Llama a la RenewOfferingoperación para cambiar la cantidad de dispositivos de tu cuenta.

Conceptos de 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).

En esta sección se describen conceptos importantes de Device Farm.

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

Para obtener más información acerca de los tipos de pruebas admitidos en Device Farm, consulte Marcos de pruebas y pruebas integradas 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 e iOS únicos y populares, y combinaciones de sistemas operativos. La lista de dispositivos disponibles crece a medida que se lanzan nuevos dispositivos al mercado. Para ver la lista completa de dispositivos, consulta la lista interactiva de dispositivos de AWS la consola.

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 Dispositivos privados en AWS Device Farm.

Marcas de dispositivos

Device Farm realiza pruebas en dispositivos móviles y tabletas físicos desde una variedad de OEMs.

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 tienen conectividad 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.

Los servicios de Google Play y Google Chrome están instalados en los dispositivos Android.

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, videos, 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 video 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 Personalización del entorno de pruebas personalizado en AWS Device Farm..

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 con la AWS CLI y Creación de 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. Estos incluyen coordenadas de latitud y longitud, datos adicionales (contenidos en un archivo.zip) y aplicaciones auxiliares (aplicaciones que se deben instalar antes de probar la aplicación). En Android, se pueden cambiar algunos ajustes adicionales, como la configuración regional y los estados de la radio (Bluetooth, GPS, NFC y Wi-Fi).

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 registros 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.

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 Creación de una ejecución de prueba en Device Farm y Ejecuciones de prueba en AWS Device Farm.

Aplicaciones en AWS Device Farm

En las siguientes secciones se ofrece información sobre los comportamientos de las aplicaciones en Device Farm.

Temas

- Instrumentación de aplicaciones
- Volver a firmar las aplicaciones en las ejecuciones
- Aplicaciones ocultas en las ejecuciones

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 ciertos derechos, como el grupo de aplicaciones, los dominios asociados, Game Center, HealthKit, HomeKit, la configuración de accesorios inalámbricos, las compras en la aplicación, el sonido en la aplicación, Apple Pay, las notificaciones push y la configuración y control de VPN.

Para aplicaciones Android, Device Farm vuelve a firmar la aplicación. Esto podría afectar a la funcionalidad que depende de la firma de la aplicación, como el API de Google Maps para Android. También podrían activarse los sistemas antipiratería y antimanipulación disponibles de productos como DexGuard.

Aplicaciones ocultas en las ejecuciones

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

Informes en AWS Device Farm

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

Temas

- 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 Visualización de informes de pruebas en Device Farm.

Sesiones de AWS Device Farm

Puede usar Device Farm para realizar pruebas interactivas de aplicaciones Android e iOS mediante sesiones de acceso remoto. Esto incluye la interacción manual en un navegador web y la ejecución de pruebas de Appium desde un cliente local en el dispositivo remoto. Los desarrolladores pueden reproducir los problemas con su aplicación o con sus pruebas de Appium en un dispositivo específico para aislarlos y resolverlos.

Temas

- Dispositivos compatibles con el acceso remoto
- Conservación de los archivos de sesión
- Instrumentación de aplicaciones
- Volver a firmar las aplicaciones en las sesiones

- Aplicaciones ocultas en las sesiones

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 video 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 refirma de una aplicación también puede activar la detección de la piratería o la manipulación en productos como los dispositivos Android. DexGuard

Aplicaciones ocultas en las sesiones

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 podrá volver a firmar la aplicación.

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

Creación de un proyecto en AWS Device Farm

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

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. Escriba un nombre para el proyecto. Si lo desea, puede proporcionar uno o varios de los parámetros siguientes y, a continuación, seleccionar Enviar.

Configuración de Virtual Private Cloud (VPC)

Seleccione una VPC, subredes y grupo de seguridad para aplicarlos al dispositivo que se está probando y a su host de prueba emparejado. Esta función solo es compatible con dispositivos privados. Para obtener más información, consulte VPC-ENI en AWS Device Farm.

Rol de ejecución (ARN)

Una función de IAM que debe asumir el ejecutor de pruebas en entornos de prueba personalizados. Para obtener más información, consulte Acceda a los recursos de AWS mediante un rol de ejecución de IAM.

Variables de entorno

Se insertarán una o más variables en el entorno del proceso del ejecutor de la prueba. Los nombres de las variables que comienzan por «DEVICEFARM_» están reservados para el uso del servicio. Recomendamos no almacenar valores confidenciales en estas variables de entorno y, en su lugar, sugerimos utilizar una función de ejecución de IAM para obtener dichos valores de AWS Secrets Manager durante la prueba.

Crear un proyecto (AWS CLI)

- Ejecute create-project especificando el nombre del proyecto.

  Ejemplo:

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

  La AWS CLI respuesta 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 AWS CLIReferencia de .

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.

Visualización de 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
- Visualización de la lista de proyectos (consola)
- Visualizar la lista de proyectos (AWS CLI)
- Visualizar la lista de proyectos (API)

Requisitos previos

- Cree al menos un proyecto en Device Farm. Siga las instrucciones de Creación de 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 AWS CLIReferencia de .

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.

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 de AWS Device Farm, AWS Command Line Interface (AWS CLI) o la API de AWS Device Farm para trabajar con las ejecuciones de prueba.

Temas

Creación de 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 Creación de 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 ejecución.
5. (Opcional) En Configuración de ejecución, en la sección Nombre de ejecución, escriba un nombre para la ejecución. Si no se proporciona ningún nombre, la consola de Device Farm asignará a la ejecución el nombre “My Device Farm run” de forma predeterminada.
6. (Opcional) En Configuración de ejecución, en la sección Tiempo de espera del trabajo, puede especificar el tiempo de espera de la ejecución de la prueba. Si utiliza un número ilimitado de ranuras de prueba, confirme que Sin medidor esté seleccionado en Método de facturación.
7. En Configuración de ejecución, en la sección Tipo de ejecución, seleccione el tipo de ejecución. Seleccione Aplicación Android si no tiene ninguna aplicación lista para probarla o si está probando una aplicación en Android (.apk). Seleccione Aplicación iOS si está probando una aplicación iOS (.ipa). Seleccione Aplicación web si desea probar aplicaciones web.
8. En Seleccionar aplicación, en la sección Opciones de selección de aplicaciones, elija Seleccionar aplicación de muestra proporcionada por Device Farm si no tiene ninguna aplicación disponible para probarla. Si va a traer su propia aplicación, seleccione Cargar aplicación propia y elija el archivo de su aplicación. Si carga una aplicación iOS, asegúrese de elegir Dispositivo iOS, en lugar de un simulador.
9. En Configurar prueba, seleccione uno de los marcos de prueba 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.

Para obtener más información acerca de los conjuntos de pruebas disponibles, consulte Marcos de pruebas y pruebas integradas en AWS Device Farm.

10. Si no ha elegido Integrado: fuzzing, seleccione Elegir archivo en Seleccionar paquete de prueba. Busque y elija 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 de AWS Device Farm.
12. Si está utilizando un entorno de pruebas personalizado, también puede hacer lo siguiente:
    - Si desea editar la especificación de prueba predeterminada en un entorno de pruebas personalizado, seleccione Editar para actualizar la especificación YAML predeterminada.
    - Si ha modificado la especificación de prueba, seleccione Guardar como nuevo para actualizarla.
    - Puede configurar las variables de entorno. Las variables que se proporcionan aquí tendrán prioridad sobre las que se puedan configurar en el proyecto principal.
13. En Seleccionar dispositivos, realice una de las siguientes acciones:
    - 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 Creación de 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.
    - Elija Seleccionar dispositivos manualmente y, a continuación, los dispositivos deseados en los que los desea ejecutar. Esta configuración no se guardará.

    Para obtener más información, consulte Compatibilidad de dispositivos en AWS Device Farm.
14. (Opcional) Para añadir una configuración adicional, abra el menú desplegable Configuración adicional. En esta sección, puede hacer una de las siguientes acciones:

- Para proporcionar un ARN de rol de ejecución o anular uno configurado en el proyecto principal, utilice el campo ARN del rol de ejecución.
- 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.
- Seleccione Habilitar grabación de video para grabar video durante la prueba.
- Seleccione Habilitar la captura de datos de rendimiento de aplicaciones para capturar datos de desempeño en el dispositivo.

> **Note**
>
> Por el momento, la configuración del estado de radio y la configuración regional del dispositivo solo está disponible en pruebas nativas de Android.

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

15. En la parte inferior de la página, elija Crear ejecución para programar la 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 pendiente situado junto a la ejecución pasará a ser el icono del resultado de la prueba.

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

Creación de una ejecución de prueba (AWS CLI)

Puede usarlo para crear una ejecución de prueba AWS CLI .

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 Creación de 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 mediante prefirmado URLs en la Guía del usuario 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 de AWS Device Farm.

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. En este ejemplo, se utiliza curl para cargar una especificación de prueba de Appium NG JavaTest:

   ```
   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, consulteAWS CLIReferencia de .

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 Creación de 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 Visualización de 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. Crear un proyecto en Device Farm. Siga las instrucciones de Creación de 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 red para ejecuciones de AWS Device Farm

Puedes usar el modelado de la red para simular las conexiones y las condiciones de la red mientras pruebas tus aplicaciones web, iOS y Android en Device Farm. Por ejemplo, puedes simular una conectividad a Internet intermitente o con pérdidas.

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 utilizas la configuración de la red, puedes 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.

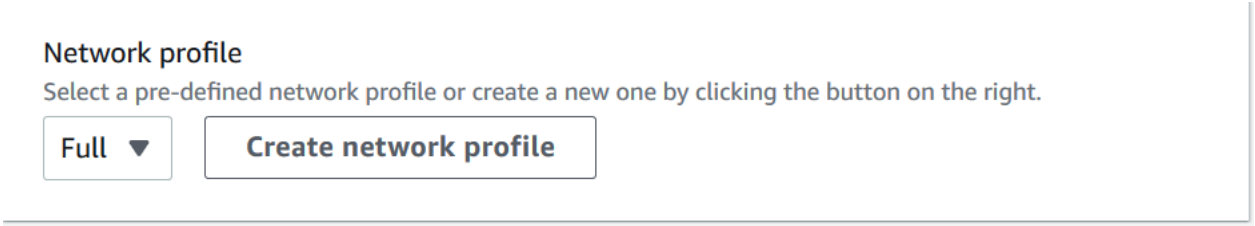
Temas

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 Creación de 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.

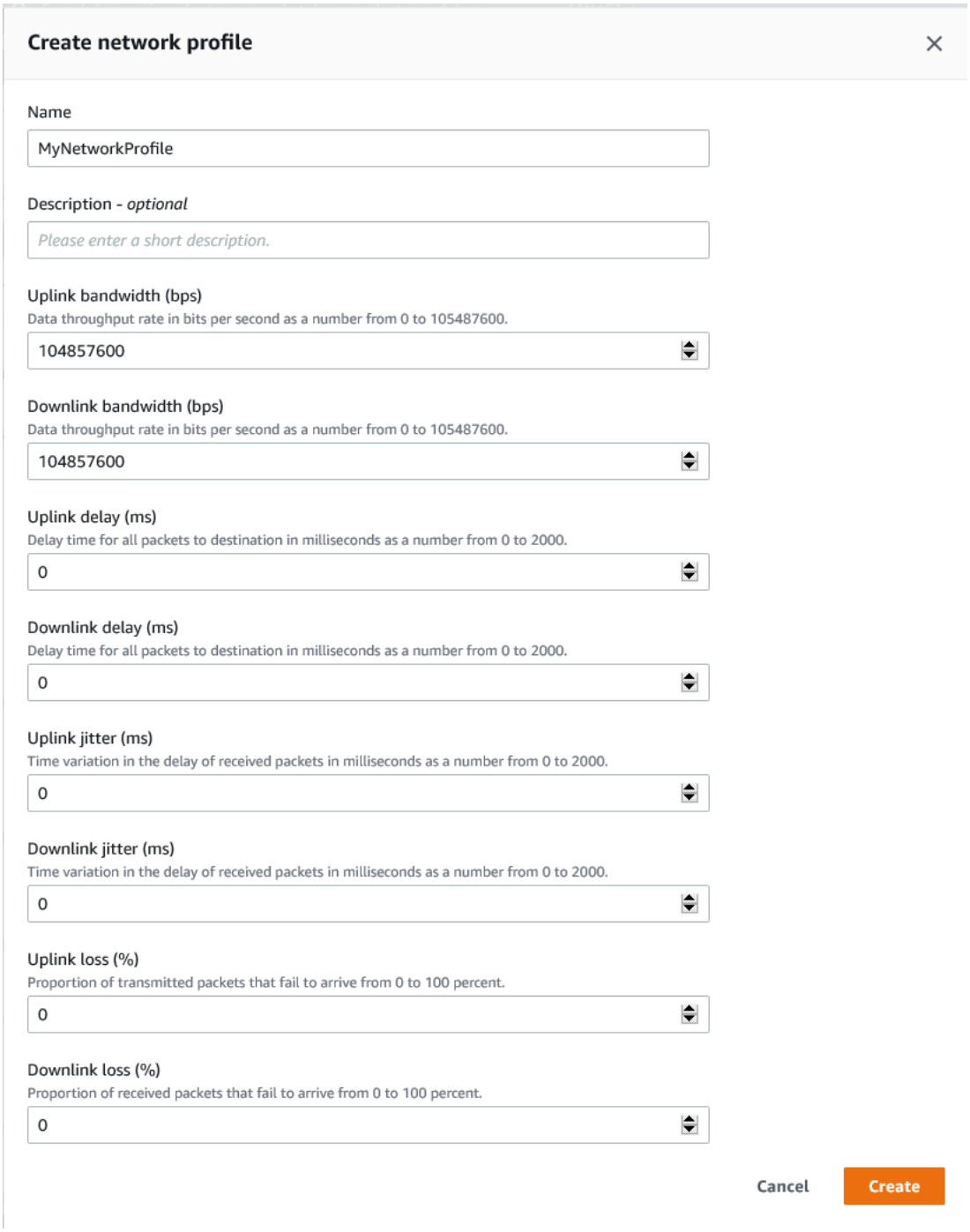


6. Seleccione Siguiente.
7. Revise y comience la ejecución de prueba.

Creación de un perfil de red

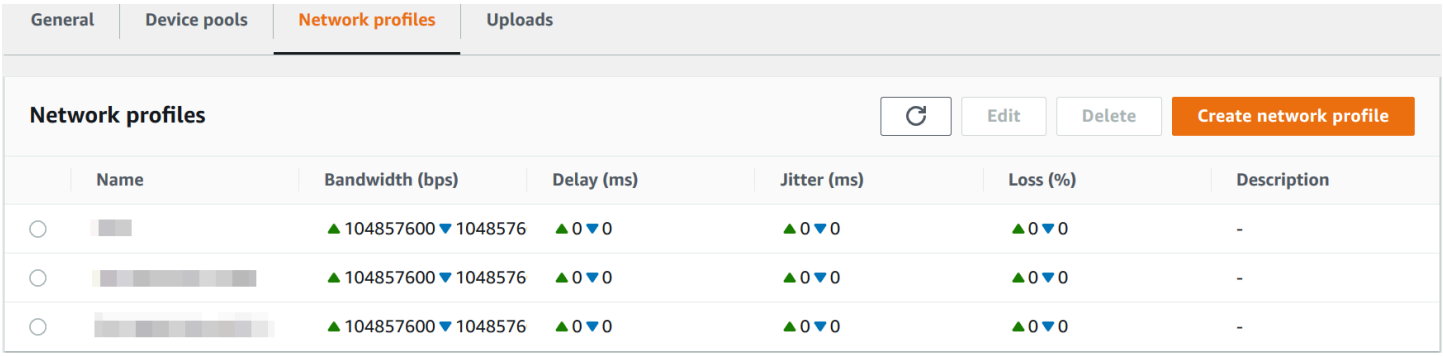
Al crear una ejecución de prueba, se puede crear un perfil de red.

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.

Detención de 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.

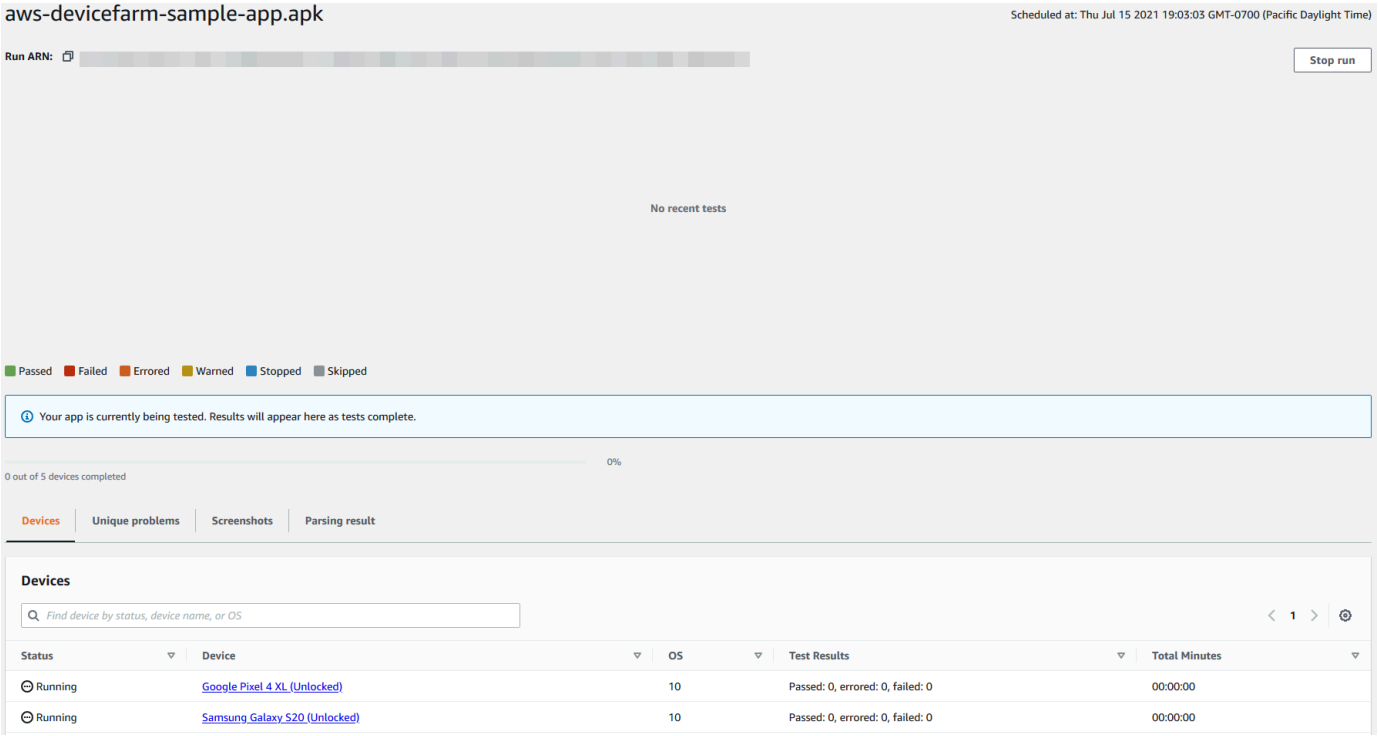
Temas

- 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.

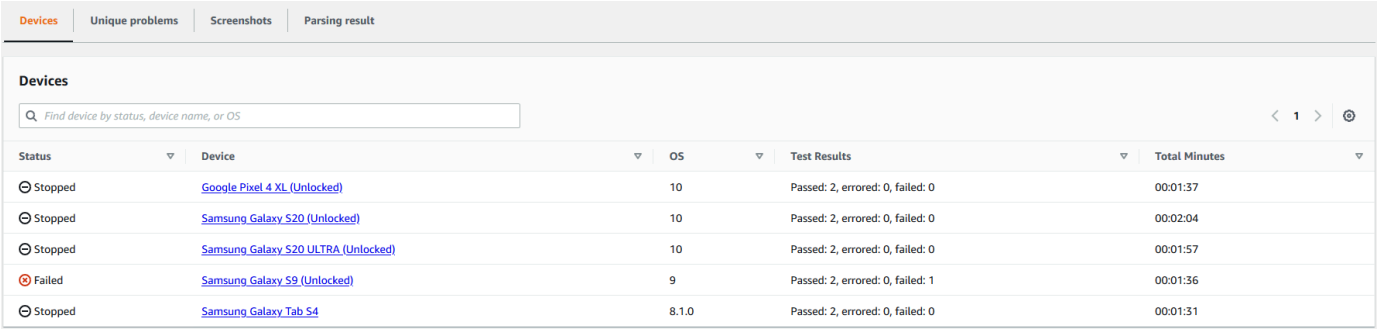


5. Seleccione Detener ejecución.

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 detener la ejecución de la prueba especificada, donde *myARN* está el nombre de recurso de Amazon (ARN) de la ejecución de la 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, consulteAWS CLIReferencia de .

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.

Visualización de 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, consulteAWS CLIReferencia de .

Visualización de una lista de ejecuciones (API)

- Llame a la API `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.

Creación de 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 Creación de 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. Elija un proyecto en la página Proyectos. En la página Detalles del proyecto, elija Configuración del proyecto. En la pestaña Grupos de dispositivos, elija 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 por factor de forma (tablet o teléfono), elija Factor de forma.
      - Para incluir los dispositivos por su estado de disponibilidad en función de la carga, elija Disponibilidad.
      - Para incluir solo dispositivos públicos o privados, elija Tipo de flota.
      - Para incluir los dispositivos por sistema operativo, elija Plataforma.
      - Algunos dispositivos tienen una etiqueta o descripción adicional acerca del dispositivo. Para buscar los dispositivos en función del contenido de sus etiquetas, seleccione Etiquetas de instancia.
      - Para incluir los dispositivos por versión del sistema operativo, elija Versión del sistema operativo.
      - Para incluir los dispositivos por modelo, elija Modelo.

   d. En Operador (segunda lista desplegable), elija una operación lógica (IGUAL QUE, CONTIENE, etc.) para incluir los dispositivos en función de la consulta. Por ejemplo, puede optar por *`Availability EQUALS AVAILABLE`* incluir los dispositivos que actualmente tienen ese `Available` estado.

   e. En Valor (tercera lista desplegable), escriba o seleccione el valor que desea especificar para los valores de Campo y Operador. Los valores están limitados en función del campo que elija. Por ejemplo, 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.

      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.

g. En Número máximo de dispositivos, introduzca la cantidad de dispositivos que desea usar en su grupo de dispositivos. Si no especifica el número máximo de dispositivos, Device Farm seleccionará todos los dispositivos de la flota que coincidan con las reglas que ha creado. Para evitar cargos adicionales, establezca este número en una cantidad que coincida con sus requisitos reales de ejecución en paralelo y variedad de dispositivos.

h. Para eliminar una etiqueta, elija Eliminar regla.

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. Si solo quiere ver las instancias de dispositivos privados, elija Ver solo las instancias de dispositivos privados.

   > **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)

> **Tip**
>
> Si no especifica el número máximo de dispositivos, Device Farm seleccionará todos los dispositivos de la flota que coincidan con las reglas que ha creado. Para evitar cargos adicionales, establezca este número en una cantidad que coincida con sus requisitos reales de ejecución en paralelo y variedad de dispositivos.

- Ejecute el comando create-device-pool.

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

Crear un grupo de dispositivos (API)

> **Tip**
>
> Si no especifica el número máximo de dispositivos, Device Farm seleccionará todos los dispositivos de la flota que coincidan con las reglas que ha creado. Para evitar cargos adicionales, establezca este número en una cantidad que coincida con sus requisitos reales de ejecución en paralelo y variedad de dispositivos.

- Llame a la API `CreateDevicePool`.

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

Análisis de los resultados de las pruebas 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. Ver los informes lo ayuda a entender qué pruebas se han superado o no, y proporciona detalles sobre el rendimiento y el comportamiento de su aplicación en las diferentes configuraciones de dispositivos.

Device Farm también recopila otros artefactos, como archivos, registros e imágenes, que puede descargar cuando la ejecución de prueba se ha completado. Esta información puede ayudarlo a analizar el comportamiento de su aplicación en dispositivos reales, identificar problemas o errores y diagnosticarlos.

Temas

- Visualización de informes de pruebas en Device Farm
- Descarga de artefactos en Device Farm

Visualización de 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 Creación de 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 pendiente situado junto a la ejecución pasará a ser el icono del resultado de la prueba. Para obtener más información, consulte Estados de los resultados de las pruebas de Device Farm.

Visualización de informes

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

Temas

- Visualización de informes de dispositivo
- Visualización de informes de conjuntos de pruebas
- Consultar los informes de pruebas
- Visualización de la información de registro para un problema, dispositivo, conjunto o prueba en un informe

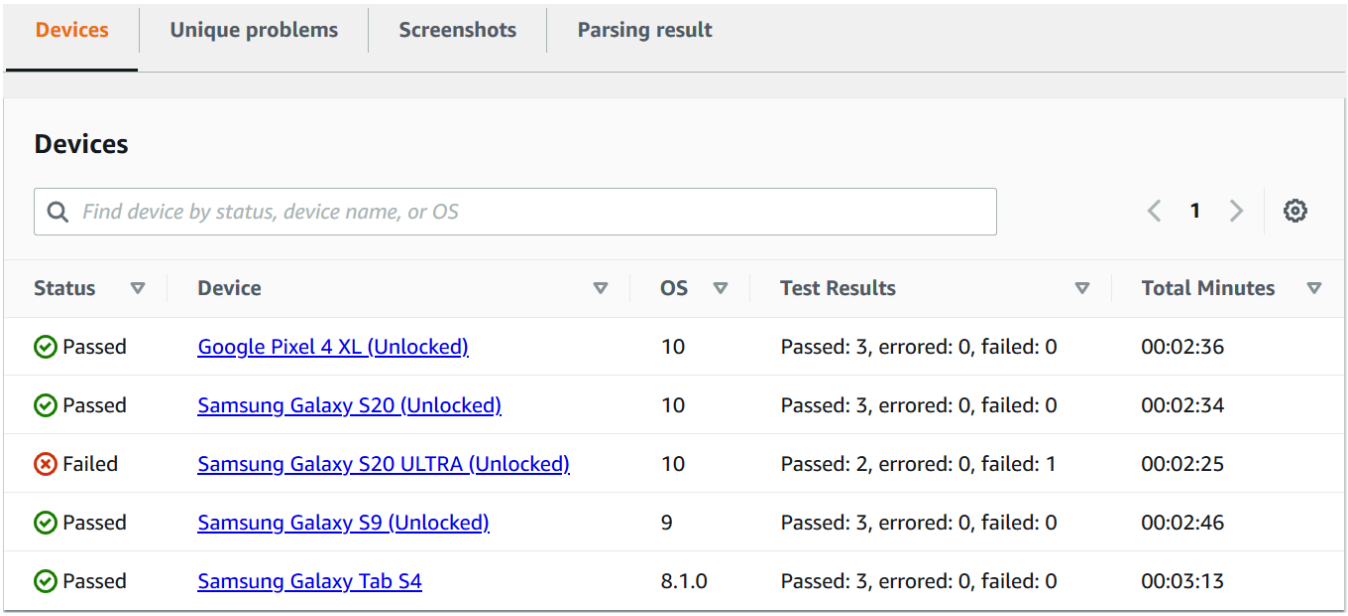
Visualización de la página de resumen de la ejecución de prueba

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 Video muestra una grabación en video 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 Estados 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 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 Video muestra una grabación en video 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 Estados 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 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 Video muestra una grabación en video 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 Estados 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 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 Video muestra una grabación en video 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 Estados 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 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 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. La hora se expresa en *`MM:SS.SSS`* formato, donde *`M`* representa los minutos y *`S`* 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
- Detenida
- 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.

Estados de los resultados de las pruebas de Device Farm

La consola de Device Farm muestra iconos que le ayudan a evaluar rápidamente el estado de la ejecución de prueba completada. Para obtener más información sobre las pruebas en Device Farm, consulte Informes en AWS Device Farm.

Temas

- Estados de una prueba individual
- Estados de varias pruebas

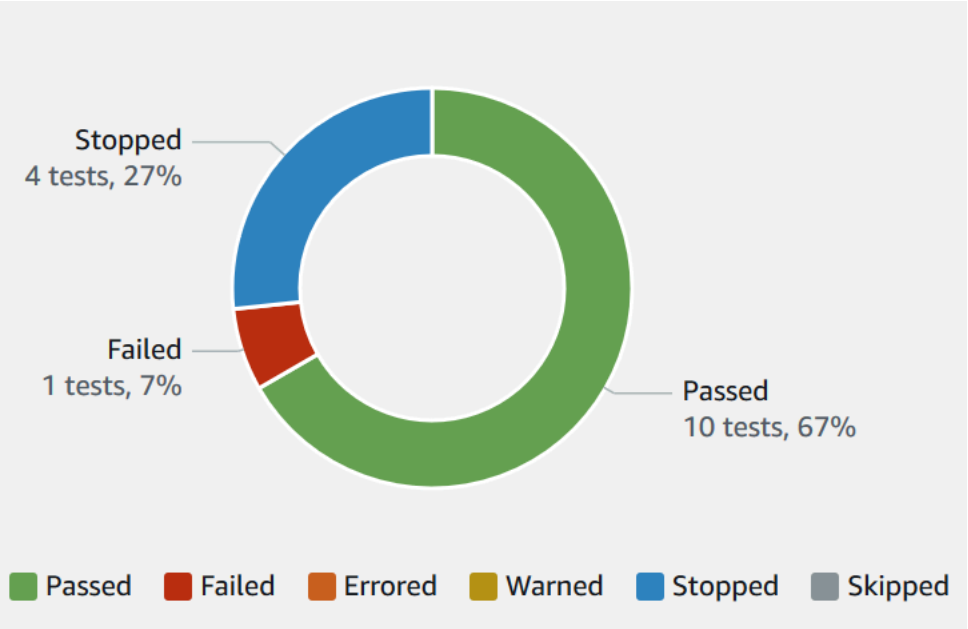
Estados de una prueba individual

Para los informes que describen una prueba individual, Device Farm muestra un icono que representa el estado del resultado de la prueba:

Description (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.	

Estados de varias pruebas

Si selecciona una ejecución finalizada, Device Farm mostrará un gráfico resumido que muestra el porcentaje de pruebas en varios estados.



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.

Descarga de 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

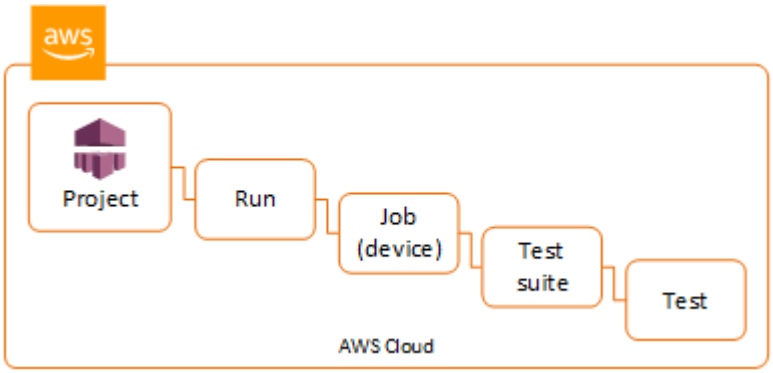
Archivos generados durante la ejecución de prueba, como los informes de Device Farm. Para obtener más información, consulte Visualización de informes de pruebas en Device Farm.

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.



Descarga 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 Descarga de artefactos en un entorno de prueba personalizado.

Descarga 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. Esta tabla muestra el ARN de entrada para cada uno de los comandos de la AWS CLI lista:

AWS CLI Comando List	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
```

Descarga 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.

Descarga de artefactos en un entorno de prueba 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.

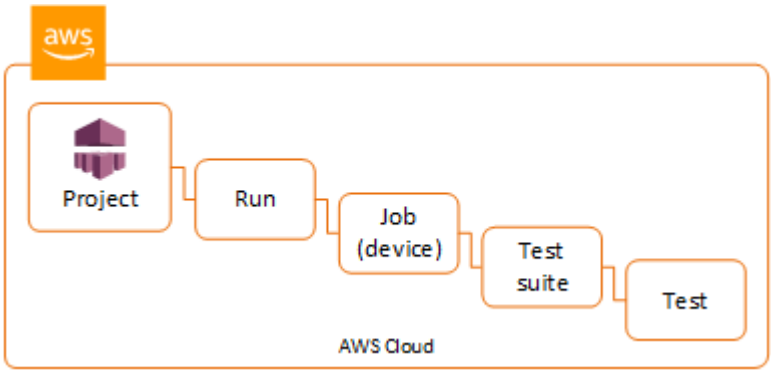
Script del shell de la especificación de prueba

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 Descarga de artefactos en Device Farm.


aws
Project
Run
Job
(device)
Test
suite
Test
AWS Cloud

Etiquetado de recursos en AWS Device Farm

AWS Device Farm funciona con la API de etiquetado AWS de Resource Groups. 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 conexión de `resourcegroupstaggingapi`. Esta operación toma una lista ARNs 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 proyectos ARNs 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 conexión `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"])
```

Marcos de pruebas y pruebas integradas 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.

Device Farm ejecuta pruebas automatizadas al hacer que subas tu aplicación y las pruebas a un bucket seguro de Amazon S3 gestionado por el servicio. Una vez cargada, activa la infraestructura subyacente, incluidos los hosts de pruebas gestionados por el servicio, y ejecuta las pruebas en paralelo en varios dispositivos. Los resultados de las pruebas se almacenan en un bucket de S3 gestionado por el servicio. Esta arquitectura se denomina ejecución del lado del servicio y es una forma rápida y eficiente de ejecutar pruebas en hosts que están físicamente cerca del dispositivo, sin necesidad de administrar usted mismo la infraestructura del host de prueba. Este enfoque se adapta bien para realizar pruebas en muchos dispositivos de forma independiente, así como para realizar pruebas desde el contexto de una CI/CD canalización.

Para obtener más información sobre cómo ejecuta las pruebas Device Farm, consulte Entornos de prueba de AWS Device Farm.

> **Note**
>
> En el caso de los evaluadores de Appium, es posible que prefiera ejecutar las pruebas de Appium desde su entorno local. Con una sesión de acceso remoto, puede ejecutar pruebas de Appium en el lado del cliente. Para obtener más información, consulta las pruebas de Appium en el lado del cliente.

Marcos de pruebas

Device Farm admite estos marcos de automatización de pruebas móviles:

Marcos de pruebas de aplicaciones Android

- Pruebas automáticas de Appium
- Instrumentación

Marcos de pruebas de aplicaciones iOS

- Pruebas automáticas de Appium
- XCTest
- XCTest INTERFAZ DE USUARIO

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 Ejecute automáticamente pruebas de Appium en Device Farm.

Marcos en un entorno de prueba personalizado

Device Farm no ofrece soporte para personalizar el entorno de prueba del XCTest marco. Para obtener más información, consulte Personalización del entorno de pruebas personalizado en AWS Device Farm..

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 de AWS Device Farm.

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)

Ejecute automáticamente pruebas de Appium en Device Farm

> **Note**
>
> En esta página, se describe la ejecución de las pruebas de Appium en el entorno de ejecución gestionado del lado del servidor de Device Farm. Para ejecutar pruebas de Appium desde su entorno local del lado del cliente durante una sesión de acceso remoto, consulte las pruebas de Appium del lado del cliente.

En esta sección, se describe cómo configurar, empaquetar y cargar las pruebas de Appium para ejecutarlas en el entorno gestionado del lado del servidor de 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.

Para obtener más información sobre las pruebas en Device Farm y cómo funciona el lado del servidor, consulte. Marcos de pruebas y pruebas integradas en AWS Device Farm

Selección de una versión de Appium

> **Note**
>
> Support para versiones específicas de Appium, controladores de Appium o programación SDKs dependerá del dispositivo y del host de prueba seleccionados para la ejecución de la prueba.

Los hosts de prueba de Device Farm vienen preinstalados con Appium para permitir una configuración más rápida de las pruebas para casos de uso más sencillos. Sin embargo, el uso del archivo de especificaciones de prueba le permite instalar diferentes versiones de Appium si es necesario.

Escenario 1: versión de Appium preconfigurada

Device Farm viene preconfigurado con diferentes versiones de servidor Appium en función del host de prueba. El host viene con herramientas que habilitan la versión preconfigurada con el controlador predeterminado de la plataforma del dispositivo (UiAutomator2 para Android y 2 para XCUITest iOS).

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

Para ver una lista del software compatible, consulte el tema en. Software compatible en entornos de prueba personalizados

Escenario 2: versión personalizada de Appium

Para seleccionar una versión personalizada de Appium, utilice el `npm` comando para instalarla. El siguiente ejemplo muestra cómo instalar la última versión de Appium 2.

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

Escenario 3: Appium en hosts iOS antiguos

En elHost de pruebas de iOS antiguo, puedes elegir versiones específicas de Appium con. `avm` Por ejemplo, para usar el `avm` comando para configurar la versión del servidor Appium`2.1.2`, agrega estos comandos a tu archivo YAML con especificaciones de prueba.

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

Selección de una WebDriverAgent versión para las pruebas de iOS

Para ejecutar las pruebas de Appium en dispositivos iOS, WebDriverAgent es necesario el uso de. Esta aplicación debe estar firmada para poder instalarse en dispositivos iOS. Device Farm proporciona versiones prefirmadas WebDriverAgent que están disponibles durante la ejecución de entornos de prueba personalizados.

El siguiente fragmento de código se puede utilizar para seleccionar una WebDriverAgent versión de Device Farm en el archivo de especificaciones de prueba que sea compatible con la versión de XCTest UI Driver.

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

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

Para obtener más información al respecto WebDriverAgent, consulte la documentación de Appium.

Integración de las pruebas de Appium con Device Farm

Siga estas instrucciones para integrar las pruebas de Appium con AWS Device Farm. Para obtener más información sobre las pruebas de Appium en Device Farm, consulte Ejecute automáticamente pruebas de Appium en Device Farm.

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 quieres instalar una versión específica de Ruby, por ejemplo*2.5.3*, ejecuta 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 ejecución.
5. En Configuración de ejecución, asigne un nombre adecuado a la prueba. Puede contener cualquier combinación de espacios o signos de puntuación.
6. Para pruebas nativas de iOS y Android

   En Configuración de ejecución, seleccione Aplicación Android si está probando una aplicación Android (.apk) o elija Aplicación iOS si está probando una aplicación iOS (.ipa). A continuación, en Seleccionar aplicación, elija Cargar aplicación propia para cargar el paquete distribuible de la 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 Configuración de ejecución, elija Aplicación web.
7. En Configurar prueba, en la sección Seleccionar marco de prueba, elija el marco de Appium con el que va a realizar las pruebas y, a continuación, Cargue su propio paquete de prueba.
8. 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.
9. Siga las instrucciones para seleccionar dispositivos e inicie la ejecución. Para obtener más información, consulte Creación de 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.

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.

Para obtener más información sobre las pruebas en Device Farm, consulte Marcos de pruebas y pruebas integradas en AWS Device Farm.

Marcos de pruebas de aplicaciones Android

Las siguientes pruebas están disponibles para dispositivos Android.

- Pruebas automáticas de Appium
- Instrumentación

Tipos de pruebas integradas para Android

Existe un tipo de prueba integrada disponible para dispositivos Android:

- Integrado: fuzzing (Android e iOS)

Instrumentación para Android y AWS Device Farm

Device Farm ofrece soporte para instrumentación (EspressoJUnit, Robotium o cualquier 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.

Para obtener más información sobre las pruebas en Device Farm, consulte Marcos de pruebas y pruebas integradas en AWS Device Farm.

Temas

¿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.

Consideraciones sobre las pruebas de instrumentación de Android

Al utilizar la instrumentación de Android, tenga en cuenta las siguientes recomendaciones y notas.

Comprobación de la compatibilidad con el sistema operativo Android

Consulte la documentación de Android para asegurarse de que la instrumentación sea compatible con su versión del sistema operativo Android.

Instalación desde la línea de comandos

Para ejecutar las pruebas de instrumentación desde la línea de comandos, consulte la Documentación de 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.Android Runner. JUnit

Grabadores de pruebas

Device Farm admite 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 métodos completos de las pruebas que coinciden con JUnit las convenciones 3 y 4. JUnit

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
```

Integración de la instrumentación de Android con Device Farm

> **Note**
>
> Siga estas instrucciones para integrar las pruebas de instrumentación de Android con AWS Device Farm. Para obtener más información sobre las pruebas de instrumentación en Device Farm, consulte Instrumentación para Android y AWS Device Farm.

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 Creación de un proyecto en AWS Device Farm.

4. Seleccione Crear regla.
5. En Seleccionar aplicación, en la sección Opciones de selección de aplicaciones, elija Cargar aplicación propia.
6. Busque y elija el archivo de aplicación de Android. El archivo debe ser un archivo .apk.
7. En la página Configurar prueba, en la sección Seleccionar marco de pruebas, elija Instrumentación y, a continuación, Elegir archivo.
8. Busque y elija el archivo .apk que contiene las pruebas.
9. Complete el resto de instrucciones para seleccionar dispositivos e inicie la ejecución.

(Opcional) Capturas de pantalla en pruebas de instrumentación de 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`

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.

Para obtener más información sobre las pruebas en Device Farm, consulte Marcos de pruebas y pruebas integradas en AWS Device Farm.

Marcos de pruebas de aplicaciones iOS

Las siguientes pruebas están disponibles para dispositivos iOS.

- Pruebas automáticas de Appium
- XCTest
- XCTest INTERFAZ DE USUARIO

Tipos de pruebas integradas para iOS

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

- Integrado: fuzzing (Android e iOS)

Integración de Device Farm con XCTest iOS

Con Device Farm, puedes usar el XCTest marco para probar tu aplicación en dispositivos reales. Para obtener más información XCTest, consulta los aspectos básicos de las pruebas con Xcode.

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

Para obtener más información sobre las pruebas en Device Farm, consulte Marcos de pruebas y pruebas integradas en AWS Device Farm.

Temas

Crea los paquetes para tu ejecución XCTest

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

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

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 XCTest paquete

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.

Sube los paquetes para tu XCTest carrera a 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 Creación de 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 ejecución.
5. En Configuración de ejecución, en la sección Tipo de ejecución, seleccione Aplicación iOS.
6. En Seleccionar aplicación, en la sección Opciones de selección de aplicaciones, elija Cargar aplicación propia. A continuación, seleccione Elegir archivo en Cargar aplicación.
7. Desplácese hasta el archivo `.ipa` de la aplicación y cárguelo.

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

8. En Configurar prueba, en la sección Seleccionar marco de prueba, elija. XCTest A continuación, seleccione Elegir archivo en Cargar aplicación.
9. Busca el `.zip` archivo que contiene el XCTest paquete de tu aplicación y cárgalo.
10. 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.
11. Seleccione Crear ejecución. Device Farm ejecuta su prueba y muestra los resultados en la consola.

Integración de la XCTest interfaz de usuario para iOS con Device Farm

Device Farm proporciona soporte para el marco de pruebas de la XCTest interfaz de usuario. En concreto, Device Farm admite pruebas de XCTest interfaz de usuario escritas tanto en Objective-C como en Swift.

El marco de la XCTest interfaz de usuario permite realizar pruebas de interfaz de usuario en el desarrollo de iOS, basadas en XCTest. Para obtener más información, consulte User Interface Testing en la iOS Developer Library.

Para obtener información general sobre las pruebas en Device Farm, consulte Marcos de pruebas y pruebas integradas en AWS Device Farm.

Sigue las instrucciones siguientes para integrar Device Farm con el marco de pruebas de XCTest interfaz de usuario para iOS.

Temas

- Prepara tus pruebas de XCTest interfaz de usuario de iOS
- Opción 1: Crear un paquete XCTest UI .ipa
- Opción 2: Crear un paquete .zip de XCTest interfaz de usuario
- Sube tus pruebas de XCTest interfaz de usuario de iOS

Prepara tus pruebas de XCTest interfaz de usuario de iOS

Puede cargar un archivo `.ipa` o `.zip` para su paquete de pruebas de XCTEST_UI.

Un archivo `.ipa` es un archivo de aplicaciones que contiene la aplicación iOS Runner en formato de paquete. No se pueden incluir archivos adicionales dentro del archivo *`.ipa`*.

Si sube un archivo `.zip`, puede contener directamente la aplicación iOS Runner o un archivo `.ipa`. También puede incluir otros archivos dentro del archivo `.zip` si desea utilizarlos durante las pruebas. Por ejemplo, puede incluir archivos como `.xctestrun`, `.xcworkspace` o `.xcodeproj` dentro de un archivo `.zip`, para ejecutar los planes de pruebas de XCUI en una granja de dispositivos. Las instrucciones detalladas sobre cómo ejecutar los planes de pruebas están disponibles en el archivo de especificaciones de pruebas predeterminado para el tipo de prueba XCUI.

Opción 1: Crear un paquete XCTest UI .ipa

El paquete yourAppNameUITest-Runner.app lo crea Xcode cuando compilas tu proyecto para probarlo. Se encuentra en el directorio Products del proyecto.

Para crear un archivo .ipa:

1. Cree un directorio denominado *`Payload`*.
2. Añada el directorio de aplicaciones al directorio Payload.
3. Archive el directorio Payload en un archivo `.zip` y, a continuación, cambie la extensión del archivo a `.ipa`.

La siguiente estructura de carpetas muestra cómo se *`my-project-nameUITest-Runner.app`* empaquetaría como un archivo una aplicación de ejemplo llamada: `.ipa`

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

Opción 2: Crear un paquete .zip de XCTest interfaz de usuario

Device Farm genera automáticamente un `.xctestrun` archivo para ejecutar todo el conjunto de pruebas de XCTest interfaz de usuario. Si quiere usar su propio archivo `.xctestrun` en Device

Farm, puede comprimir sus archivos `.xctestrun` y el directorio de aplicaciones en un archivo `.zip`. Si ya tiene un `.ipa` archivo para su paquete de prueba, puede incluirlo aquí en su lugar*`*-Runner.app`*.

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

Si quieres ejecutar un plan de pruebas de Xcode para tus pruebas de XCUI en Device Farm, puedes crear un zip que contenga el my-project-nameUITestarchivo.app o my-project-name UITest.ipa de Runner.app y los archivos de código fuente de xcode necesarios para ejecutar XCTEST_UI con planes de pruebas, incluido un archivo o. `.xcworkspace` `.xcodeproj`

A continuación, se muestra un ejemplo de zip con un archivo `.xcodeproj`:

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

A continuación, se muestra un ejemplo de zip con un archivo `.xcworkspace`:

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

```
    ### SampleWorkspace.xcworkspace
        ### contents.xcworkspacedata
```

> **Note**
>
> Asegúrese de no tener un directorio llamado «Payload» dentro del paquete .zip de la interfaz de usuario. XCTest

Sube tus pruebas de XCTest interfaz de usuario de 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 Creación de un proyecto en AWS Device Farm.

4. Seleccione Crear ejecución.
5. En Configuración de ejecución, en la sección Tipo de ejecución, seleccione Aplicación iOS.
6. En Seleccionar aplicación, en la sección Opciones de selección de aplicaciones, elija Cargar aplicación propia. A continuación, seleccione Elegir archivo en Cargar aplicación.
7. 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.

8. En Configurar la prueba, en la sección Seleccionar el marco de prueba, elija UI. XCTest A continuación, seleccione Elegir archivo en Cargar aplicación.

9. Busca y selecciona el archivo.ipa o .zip que contiene tu ejecutor de pruebas de XCTest interfaz de usuario de iOS.
10. Complete los demás pasos del proceso de creación de la ejecución. Seleccione los dispositivos en los que desea realizar la prueba y, de manera opcional, puede especificar una configuración adicional.
11. Seleccione Crear ejecución. Device Farm ejecuta su prueba y muestra los resultados en la consola.

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 “Pruebas automáticas de Appium”

Para obtener más información sobre las pruebas en Device Farm, consulte Marcos de pruebas y pruebas integradas en AWS Device Farm.

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.

Pruebas integradas en AWS Device Farm

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

Las pruebas integradas permiten probar la aplicación en varios dispositivos sin tener que escribir ni mantener scripts de automatización de pruebas. Esto puede ahorrarle tiempo y esfuerzo, especialmente cuando esté empezando a usar Device Farm. Device Farm ofrece el siguiente tipo de prueba integrada:

- Integrado: fuzzing (Android e iOS): 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.

Para obtener más información sobre las pruebas y los marcos de pruebas en Device Farm, consulte Marcos de pruebas y pruebas integradas en AWS Device Farm.

Ejecución de la prueba de difusión integrada de Device Farm (Android e iOS)

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

Para obtener más información sobre las pruebas en Device Farm, consulte Marcos de pruebas y pruebas integradas en AWS Device Farm.

Cómo 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 Creación de un proyecto en AWS Device Farm.

4. Seleccione Crear ejecución.
5. En Configuración de ejecución, en la sección Tipo de ejecución, seleccione el tipo de ejecución. Seleccione Aplicación Android si no tiene ninguna aplicación lista para probarla o si está probando una aplicación en Android (.apk). Seleccione Aplicación iOS si está probando una aplicación iOS (.ipa).
6. En Seleccionar aplicación, elija Seleccionar aplicación de muestra proporcionada por Device Farm si no tiene ninguna aplicación disponible para probarla. Si va a traer su propia aplicación, seleccione Cargar aplicación propia y elija el archivo de su aplicación.
7. En la página Configurar prueba, en la sección Seleccionar marco de pruebas, elija Integrado: fuzzing.
8. 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.
9. Complete el resto de instrucciones para seleccionar dispositivos e inicie la ejecución.

Personalización del entorno de pruebas personalizado 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 preprocessing/postprocessing sobrecarga.
- Registro y transmisión de video en directo: los registros de pruebas y el video 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 con la AWS CLI y Creación de una ejecución de prueba en Device Farm.

Temas

Pruebe la referencia y la sintaxis de las especificaciones

La especificación de prueba (especificación de prueba) es un archivo que se utiliza para definir entornos de prueba personalizados en Device Farm.

Flujo de trabajo de especificaciones de prueba

La especificación de prueba de Device Farm ejecuta las fases y sus comandos en un orden predeterminado, lo que le permite personalizar la forma en que se prepara y ejecuta su entorno. Cuando se ejecuta cada fase, sus comandos se ejecutan en el orden indicado en el archivo de especificaciones de la prueba. Las fases se ejecutan en la siguiente secuencia

1. `install`- Aquí es donde se deben definir acciones como descargar, instalar y configurar las herramientas.
2. `pre_test`- Aquí es donde deben definirse las acciones previas a las pruebas, como iniciar procesos en segundo plano.
3. `test`- Aquí es donde debe definirse el comando que invoca la prueba.
4. `post_test`- Aquí es donde se deben definir las tareas finales que deban ejecutarse una vez finalizada la prueba, como la generación del informe de la prueba y la agregación de archivos de artefactos.

Sintaxis de la especificación de prueba

El siguiente es el esquema YAML para un archivo de especificaciones de prueba

```
version: 0.1
```

```
android_test_host: "string"
ios_test_host: "string"

phases:
  install:
    commands:
      - "string"
      - "string"
  pre_test:
    commands:
      - "string"
      - "string"
  test:
    commands:
      - "string"
      - "string"
  post_test:
    commands:
      - "string"
      - "string"

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

`version`

(Obligatorio, número)

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

`android_test_host`

(Opcional, cadena)

El host de prueba que se seleccionará para las pruebas que se realicen en dispositivos Android. Este campo es obligatorio para las pruebas realizadas en dispositivos Android. Para obtener más información, consulte Hosts de prueba disponibles para entornos de prueba personalizados.

`ios_test_host`

(Opcional, cadena)

El host de prueba que se seleccionará para las ejecuciones de prueba realizadas en dispositivos iOS. Este campo es obligatorio para las pruebas realizadas en dispositivos iOS con una versión principal superior a la 26. Para obtener más información, consulte Hosts de prueba disponibles para entornos de prueba personalizados.

`phases`

Esta sección contiene grupos de comandos ejecutados durante una ejecución de prueba, donde cada fase es opcional. Los nombres de las fases de prueba permitidos son: `install`,`pre_test`, `test`, y`post_test`.

- `install`- Las dependencias predeterminadas para 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`- Los comandos, si los hubiera, se ejecutaron antes de la prueba automatizada.
- `test`- Los comandos ejecutados durante la ejecución de la prueba automatizada. Si algún comando de la fase de prueba falla (es decir, devuelve un código de salida distinto de cero), la prueba se marca como fallida
- `post_test`- Los comandos, si los hay, se ejecutan después de la ejecución de la prueba automatizada. Esto se ejecutará independientemente de que la prueba de la `test` fase tenga éxito o no.

`commands`

(Opcional, lista [cadena])

Una lista de cadenas para ejecutar como un comando de shell durante la fase.

`artifacts`

(Opcional, lista [cadena])

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 Descarga de artefactos en un entorno de prueba 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.

Ejemplos de especificaciones de prueba

Los siguientes ejemplos muestran las especificaciones de prueba que se pueden ejecutar en Device Farm.

Simple Demo

El siguiente es un ejemplo de archivo de especificaciones de prueba que simplemente se registra `Hello world!` como un artefacto de ejecución de prueba.

```
version: 0.1

android_test_host: amazon_linux_2
ios_test_host: macos_sequoia

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

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

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

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

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

Appium Android

El siguiente es un ejemplo de archivo de especificaciones de prueba que configura una ejecución de prueba de Appium Java TestNG en Android.

```
version: 0.1

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

phases:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Appium iOS

El siguiente es un ejemplo de archivo de especificaciones de prueba que configura una ejecución de prueba de Appium Java TestNG en iOS.

```
version: 0.1

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

phases:

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

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

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

      # When running iOS tests with Appium version 2, the XCUITest driver is
 preinstalled using driver
```

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

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

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

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

     # Device Farm provides multiple pre-built versions of WebDriverAgent (WDA), a
required
     # Appium dependency for iOS, where each version corresponds to the XCUITest
driver version selected.
     # If Device Farm cannot find a corresponding version of WDA for your XCUITest
driver,
     # the latest available version is selected by default.
     - |-
       APPIUM_DRIVER_VERSION=$(appium driver list --installed --json | jq -r
".xcuitest.version" | cut -d "." -f 1);
       CORRESPONDING_APPIUM_WDA=$(env | grep
"DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V${APPIUM_DRIVER_VERSION}")
```

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

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

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

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

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

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

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

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

     # - echo "Unzipping the tests JAR file"
```

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

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

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

Appium (Both Platforms)

El siguiente es un ejemplo de archivo de especificaciones de prueba que configura una ejecución de prueba de Appium Java TestNG tanto en Android como en iOS.

```
version: 0.1

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

phases:

  # The install phase contains commands for installing dependencies to run your
 tests.
```

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

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

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

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

      # When running iOS tests with Appium version 2, the XCUITest driver is
 preinstalled using driver
```

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

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

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

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

     # Device Farm provides multiple pre-built versions of WebDriverAgent (WDA), a
required
     # Appium dependency for iOS, where each version corresponds to the XCUITest
driver version selected.
     # If Device Farm cannot find a corresponding version of WDA for your XCUITest
driver,
     # the latest available version is selected by default.
     - |-
       if [ $DEVICEFARM_DEVICE_PLATFORM_NAME = "iOS" ]; then
         APPIUM_DRIVER_VERSION=$(appium driver list --installed --json | jq -r
".xcuitest.version" | cut -d "." -f 1);
         CORRESPONDING_APPIUM_WDA=$(env | grep
"DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V${APPIUM_DRIVER_VERSION}")
```

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

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

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

```
            \"platformName\": \"$DEVICEFARM_DEVICE_PLATFORM_NAME\", \
            \"appium:udid\":\"$DEVICEFARM_DEVICE_UDID\", \
            \"appium:platformVersion\": \"$DEVICEFARM_DEVICE_OS_VERSION\", \
            \"appium:chromedriverExecutableDir\":
\"$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR\", \
            \"appium:automationName\": \"UiAutomator2\"}" \
            >> $DEVICEFARM_LOG_DIR/appium.log 2>&1 &
        else
          appium --base-path=$APPIUM_BASE_PATH --log-timestamp \
            --log-no-colors --relaxed-security --default-capabilities \
            "{\"appium:deviceName\": \"$DEVICEFARM_DEVICE_NAME\", \
            \"platformName\": \"$DEVICEFARM_DEVICE_PLATFORM_NAME\", \
            \"appium:udid\":\"$DEVICEFARM_DEVICE_UDID_FOR_APPIUM\", \
            \"appium:platformVersion\": \"$DEVICEFARM_DEVICE_OS_VERSION\", \
            \"appium:derivedDataPath\": \"$DEVICEFARM_WDA_DERIVED_DATA_PATH\", \
            \"appium:usePrebuiltWDA\": true, \
            \"appium:automationName\": \"XCUITest\"}" \
            >> $DEVICEFARM_LOG_DIR/appium.log 2>&1 &
        fi;

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

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

      # The following command runs your Appium Java TestNG test.
```

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

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

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

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

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

Hosts para entornos de prueba personalizados

Device Farm admite un conjunto de sistemas operativos con software preconfigurado mediante el uso de un entorno host de prueba. Durante la ejecución de la prueba, Device Farm utiliza instancias gestionadas por Amazon (hosts) que se conectan dinámicamente al dispositivo seleccionado que se está probando. Esta instancia se limpia por completo y no se reutiliza entre ejecuciones, y se termina con los artefactos generados una vez finalizada la ejecución de la prueba.

Temas

Hosts de prueba disponibles para entornos de prueba personalizados

Device Farm administra completamente los hosts de prueba. En la siguiente tabla se enumeran los hosts de prueba de Device Farm actualmente disponibles y compatibles para entornos de prueba personalizados.

Plataforma de dispositivos	Host de prueba	Sistema operativo	Arquitectura (es)	Dispositivos admitidos
Android	amazon_linux_2	Amazon Linux 2	x86_64	Android6 y superior
iOS	macos_sequoia	macOS Sequoia(versión 15)	arm64	iOS15 a 26

> **Note**
>
> Periódicamente, Device Farm agrega nuevos hosts de prueba para una plataforma de dispositivos a fin de admitir las versiones más recientes del sistema operativo del dispositivo

y sus dependencias. Cuando esto ocurre, los hosts de prueba más antiguos para la plataforma de dispositivo correspondiente están sujetos a la finalización del soporte.

Versión del sistema operativo

Cada host de prueba disponible utiliza una versión específica del sistema operativo compatible con Device Farm en ese momento. Aunque intentamos tener la última versión del sistema operativo, puede que no sea la última versión de distribución pública disponible. Device Farm actualizará periódicamente el sistema operativo con actualizaciones de versiones menores y parches de seguridad.

Para conocer la versión específica (incluida la versión secundaria) del sistema operativo que se utiliza durante la ejecución de la prueba, puede añadir el siguiente fragmento de código a cualquiera de las fases del archivo de especificaciones de la prueba.

Example

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

Selección de un host de pruebas para entornos de prueba personalizados

Puede especificar el host de pruebas de Android e iOS en las `ios_test_host` variables correspondientes `android_test_host` de su archivo de especificaciones de prueba.

Si no especifica una selección de host de prueba para la plataforma de dispositivo determinada, las pruebas se ejecutarán en el host de prueba que Device Farm haya establecido como predeterminado para el dispositivo y la configuración de prueba especificados.

> **Important**
>
> Al realizar las pruebas en iOS 18 y versiones anteriores, se utilizará un host de prueba antiguo cuando no se seleccione ningún anfitrión. Para obtener más información, consulta el tema sobreHost de pruebas de iOS antiguo.

Como ejemplo, revise el siguiente fragmento de código:

Example

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

phases:
  # ...
```

Software compatible en entornos de prueba personalizados

Device Farm utiliza máquinas host que vienen preinstaladas con muchas de las bibliotecas de software necesarias para ejecutar los marcos de pruebas compatibles con nuestro servicio, lo que proporciona un entorno de pruebas listo para el lanzamiento. Device Farm admite varios idiomas mediante el uso de nuestro mecanismo de selección de software y actualizará periódicamente las versiones de los idiomas incluidos en el entorno.

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 Ejemplos de especificaciones de prueba.

Software preconfigurado

Para facilitar las pruebas de los dispositivos en cada plataforma, se proporcionan las siguientes herramientas en el host de prueba:

Tools (Herramientas)	Plataforma (s) de dispositivos
Android SDK Build-Tools	Android

Tools (Herramientas)	Plataforma (s) de dispositivos
Android SDK Platform-Tools(incluyeadb)	Android
Xcode	iOS

Software seleccionable

Además del software preconfigurado en el host, Device Farm ofrece una forma de seleccionar determinadas versiones del software compatible mediante las `devicefarm-cli` herramientas.

La siguiente tabla contiene el software seleccionable y los hosts de prueba que lo contienen.

Software/herramienta	Hosts que admiten este software	Comando para usar en la especificación de prueba
Java 17	amazon_linux_2 macos_sequoia	`devicefarm-cli use java 17`
Java 11	amazon_linux_2 macos_sequoia	`devicefarm-cli use java 11`
Java 8	amazon_linux_2 macos_sequoia	`devicefarm-cli use java 8`
Node.js 20	amazon_linux_2 macos_sequoia	`devicefarm-cli use node 20`
Node.js 18	amazon_linux_2 macos_sequoia	`devicefarm-cli use node 18`
Node.js 16	amazon_linux_2	`devicefarm-cli use node 16`

Software/herramienta	Hosts que admiten este software	Comando para usar en la especificación de prueba
Python 3.11	amazon_linux_2 macos_sequoia	`devicefarm-cli use python 3.11`
Python 3.10	amazon_linux_2 macos_sequoia	`devicefarm-cli use python 3.10`
Python 3.9	amazon_linux_2 macos_sequoia	`devicefarm-cli use python 3.9`
Python 3.8	amazon_linux_2	`devicefarm-cli use python 3.8`
Ruby 3.2	amazon_linux_2 macos_sequoia	`devicefarm-cli use ruby 3.2`
Ruby 2.7	amazon_linux_2	`devicefarm-cli use ruby 2.7`
Appium 3	amazon_linux_2	`devicefarm-cli use appium 3`
Appium 2	amazon_linux_2 macos_sequoia	`devicefarm-cli use appium 2`
Appium 1	amazon_linux_2	`devicefarm-cli use appium 1`
Xcode 26	macos_sequoia	`devicefarm-cli use xcode 26`
Xcode 16	macos_sequoia	`devicefarm-cli use xcode 16`

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

Uso de la herramienta devicefarm-cli en entornos de prueba personalizados

El anfitrión de la prueba utiliza una herramienta de administración de versiones estandarizada llamada `devicefarm-cli` para seleccionar las versiones de software. Esta herramienta es independiente del host de pruebas de Device Farm 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.

> **Important**
>
> Esta herramienta de línea de comandos no está disponible en los hosts iOS antiguos. Para obtener más información, consulte el tema sobreHost de pruebas de iOS antiguo.

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 la versión de Appium de a: *1 2*

```
$ 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.

Para obtener más información sobre el software preinstalado en el host de prueba, consulte. Software compatible en entornos de prueba personalizados

Entorno de pruebas para dispositivos 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 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 video 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

Intervalos de IP admitidos para el entorno de pruebas de Amazon Linux 2 en Device Farm

Los clientes necesitan saber el rango de IP desde el que se origina el tráfico de Device Farm, especialmente para configurar sus firewalls y los ajustes de seguridad. En el caso de los hosts de prueba de Amazon EC2, el rango de IP abarca toda la región `us-west-2`. Para los hosts de prueba de Amazon Linux 2, que es la opción predeterminada para las nuevas ejecuciones de Android, se han restringido los rangos. El tráfico ahora se origina en un conjunto específico de puertas de enlace NAT, lo que restringe el rango de IP a las siguientes direcciones:

Rangos de IP
44.236.137.143
52,13,151,244
52,35189,191
54,201,250,26

Para obtener más información acerca de los entornos de pruebas de Android en Device Farm, consulte Entorno de pruebas para dispositivos Android.

Entorno de pruebas para dispositivos iOS

Device Farm utiliza instancias macOS (hosts) gestionadas por Amazon que se conectan dinámicamente al dispositivo iOS durante la ejecución de la prueba. Cada host viene preconfigurado con un software que permite realizar pruebas de dispositivos en varias plataformas de prueba populares, como XCTest UI y Appium.

La versión actual del host de pruebas de iOS ha mejorado la experiencia de prueba en comparación con las versiones anteriores, incluidas las siguientes:

- Experiencia uniforme con el sistema operativo anfitrión y las herramientas para iOS 15 e iOS 26 Antes, el anfitrión de la prueba lo determinaba el dispositivo en uso, lo que generaba un entorno de software fragmentado cuando se ejecutaba en varias versiones de iOS. La experiencia actual permite seleccionar el host de forma sencilla para crear un entorno uniforme en todos los dispositivos. Esto permitirá que la misma versión y herramientas de macOS (como Xcode) estén disponibles en todos los dispositivos iOS.
- Mejoras de rendimiento en las pruebas de iOS 15 y 16 Gracias a la infraestructura actualizada, el tiempo de configuración ha mejorado considerablemente en las pruebas de iOS 15 y 16.
- Versiones de software seleccionables estandarizadas para las dependencias compatibles Ahora tenemos el sistema de selección de `devicefarm-cli` software en los hosts de prueba de iOS y Android, lo que le permite seleccionar la versión que prefiera de nuestras dependencias compatibles. Para las dependencias compatibles (como Java, Python, Node.js, Ruby y Appium), las versiones se podrán seleccionar mediante la especificación de prueba. Para hacerse una idea de cómo funciona esta función, consulte el tema sobre. Software compatible en entornos de prueba personalizados

> **Important**
>
> Si se ejecuta en iOS 18 o versiones anteriores, las pruebas se ejecutarán en los hosts de prueba antiguos de forma predeterminada. Consulta el tema siguiente sobre cómo migrar desde los hosts antiguos.

Host de pruebas de iOS antiguo

Para las pruebas existentes en iOS 18 y versiones anteriores, los anfitriones de pruebas antiguos se seleccionan de forma predeterminada para los entornos de prueba personalizados. La siguiente tabla contiene la versión del host de prueba con la que se ejecuta la versión del dispositivo iOS.

Sistema operativo	Arquitecturas	Predeterminado para los dispositivos
macOS Sonoma(versión 14)	arm64	iOS 18
macOS Ventura(versión 13)	arm64	iOS 17
macOS Monterey(versión 12)	x86_64	iOS 16y más abajo

Para seleccionar los hosts de prueba más nuevos, consulte el tema correspondienteMigración de sus entornos de prueba personalizados a los nuevos hosts de prueba de iOS.

Software compatible para dispositivos iOS

Para permitir las pruebas de dispositivos iOS, los hosts de prueba de Device Farm para dispositivos iOS vienen preconfigurados con Xcode y sus herramientas de línea de comandos asociadas. Para ver otro software disponible, consulta el tema correspondiente. Software compatible en entornos de prueba personalizados

Migración de sus entornos de prueba personalizados a los nuevos hosts de prueba de iOS

Para migrar las pruebas existentes del host anterior al nuevo host de pruebas de macOS, tendrás que desarrollar nuevos archivos de especificaciones de prueba basados en los ya existentes.

El enfoque recomendado es empezar con el archivo de especificaciones de prueba de ejemplo para los tipos de prueba que desees y, a continuación, migrar los comandos pertinentes del archivo de especificaciones de prueba anterior al nuevo. Esto le permite aprovechar las nuevas funciones y optimizaciones de la especificación de prueba del ejemplo para el nuevo host y, al mismo tiempo, reutilizar fragmentos de código existentes.

Temas

- Tutorial: Migración de archivos de especificaciones de prueba de iOS con la consola

- Diferencias entre los hosts de prueba nuevos y antiguos

Tutorial: Migración de archivos de especificaciones de prueba de iOS con la consola

En este ejemplo, la consola Device Farm se usará para incorporar una especificación de prueba de un dispositivo iOS existente para usar el nuevo host de prueba.

Paso 1: Crear un nuevo archivo de especificaciones de prueba con la consola

1. Inicie sesión en la consola de AWS Device Farm.
2. Navegue hasta el proyecto Device Farm que contiene sus pruebas de automatización.
3. Descargue una copia de la especificación de prueba existente con la que desee incorporarla.
    a. Haz clic en la opción «Configuración del proyecto» y dirígete a la pestaña Cargas.
    b. Navega hasta el archivo de especificaciones de la prueba con el que deseas incorporarla.
    c. Haga clic en el botón Descargar para hacer una copia local de este archivo.
4. Vuelva a la página del proyecto y haga clic en Crear ejecución.
5. Complete las opciones del asistente como si fuera a iniciar una nueva ejecución, pero deténgase en la opción Seleccionar especificación de prueba.
6. Con la especificación de prueba de iOS seleccionada de forma predeterminada, haz clic en el botón Crear una especificación de prueba.
7. Modifique la especificación de prueba que se seleccionó de forma predeterminada en el editor de texto.
    a. Si aún no está presente, modifique el archivo de especificaciones de la prueba para seleccionar el nuevo host mediante:

        ```
        ios_test_host: macos_sequoia
        ```

    b. De la copia de las especificaciones de prueba descargada en un paso anterior, revise cada una de ellas. `phase`
    c. Copie los comandos de las fases de la especificación de prueba anterior en cada fase correspondiente de la nueva especificación de prueba, ignorando los comandos relacionados con la instalación o la selección de Java, Python, Node.js, Ruby, Appium o Xcode.
8. Introduzca un nombre de archivo nuevo en el cuadro de texto Guardar como.

9. Haga clic en el botón Guardar como nuevo para guardar los cambios.

Para ver un ejemplo de un archivo de especificaciones de prueba que puede utilizar como referencia, consulte el ejemplo que se proporciona enEjemplos de especificaciones de prueba.

Paso 2: Selección del software (software preinstalado)

En el nuevo host de prueba, las versiones de software preinstaladas se seleccionan mediante una nueva herramienta estandarizada de administración de versiones denominada. `devicefarm-cli` Estas herramientas son ahora el enfoque recomendado para usar los distintos programas que ofrecemos en los hosts de prueba.

Como ejemplo, añadiría la siguiente línea para usar un JDK 17 diferente en su entorno de prueba:

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

Para obtener más información sobre el software compatible disponible, consulte:Software compatible en entornos de prueba personalizados.

Paso 3: Uso de Appium y sus dependencias a través de las herramientas de selección de software

El nuevo host de prueba solo es compatible con Appium 2.x y versiones posteriores. Seleccione explícitamente la versión de Appium utilizando la y, al mismo tiempo`devicefarm-cli`, elimine las herramientas antiguas, como. `avm` Por ejemplo:

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

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

La versión de Appium seleccionada `devicefarm-cli` viene preinstalada con una versión compatible del controlador XCUITest para iOS.

Además, tendrás que actualizar las especificaciones de prueba para utilizarlas en lugar de. `DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V9` `DEVICEFARM_WDA_DERIVED_DATA_PATH` La nueva variable de entorno apunta a una versión prediseñada de WebDriverAgent 9.x, que es la última versión compatible con las pruebas de Appium 2.

Para obtener más información, consulte y. Selección de una WebDriverAgent versión para las pruebas de iOS Variables de entorno para las pruebas de Appium

Diferencias entre los hosts de prueba nuevos y antiguos

Al editar el archivo de especificaciones de las pruebas para usar el nuevo host de pruebas de iOS y al realizar la transición de las pruebas desde el host de pruebas anterior, ten en cuenta estas diferencias clave del entorno:

- Versiones de Xcode: en el entorno de host de pruebas antiguo, la versión de Xcode disponible se basaba en la versión iOS del dispositivo utilizada para las pruebas. Por ejemplo, las pruebas en dispositivos iOS 18 usaron Xcode 16 en el host anterior, mientras que las pruebas en iOS 17 usaron Xcode 15. En el nuevo entorno anfitrión, todos los dispositivos pueden acceder a las mismas versiones de Xcode, lo que permite disponer de un entorno uniforme para las pruebas en dispositivos con diferentes versiones. Para obtener una lista de las versiones de Xcode disponibles actualmente, consulte. Software compatible
- Selección de versiones de software: en muchos casos, las versiones de software predeterminadas han cambiado, por lo que si antes no seleccionaba explícitamente su versión de software en el host de prueba anterior, puede que desee especificarla ahora en el nuevo host de prueba mediante `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, `devicefarm-cli` tendrá una experiencia predecible y coherente con ella 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 agregado 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.
- El **`libimobiledevice`** conjunto de herramientas se ha eliminado en favor de herramientas más nuevas o propias para realizar un seguimiento de las pruebas actuales de los dispositivos iOS y los estándares de la industria. Para iOS 17 y versiones posteriores, puedes migrar la mayoría de los comandos para usar herramientas de Xcode similares, llamadas. `devicectl` Para obtener información al respecto`devicectl`, puedes ejecutarlo `xcrun devicectl help` desde una máquina con Xcode instalado.

- Las rutas de archivos que están codificadas en el archivo de especificaciones de prueba del host anterior como rutas absolutas probablemente no funcionen como se espera en el nuevo host de prueba y, 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. Para obtener más información, consulte el tema sobre. Mejores prácticas para la ejecución de entornos de pruebas personalizados
- Versión y arquitectura del sistema operativo: los hosts de prueba antiguos utilizaban diversas versiones de macOS y arquitecturas de CPU basadas en el dispositivo asignado. Como resultado, es posible que los usuarios observen algunas diferencias en las bibliotecas de sistemas disponibles en el entorno. Para obtener más información sobre la versión anterior del sistema operativo anfitrión, consulteHost de pruebas de iOS antiguo.
- Para los usuarios de Appium, la forma de seleccionar el prefijo WebDriverAgent ha cambiado a utilizar el prefijo de la variable de entorno `DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V` en lugar del prefijo anterior. `DEVICEFARM_WDA_DERIVED_DATA_PATH_V` Para obtener más información sobre la variable actualizada, consulte. Variables de entorno para las pruebas de Appium
- 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.

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.

Acceda a los recursos de AWS mediante un rol de ejecución de IAM

Device Farm admite la especificación de una función de IAM que asumirá el entorno de ejecución de la prueba personalizado durante la ejecución de la prueba. Esta función permite que sus pruebas accedan de forma segura a los recursos de AWS de su cuenta, como los buckets de Amazon S3, las tablas de DynamoDB u otros servicios de AWS de los que dependa su aplicación.

Temas

Descripción general de

Al especificar una función de ejecución de IAM, Device Farm asume esta función durante la ejecución de la prueba, lo que permite que las pruebas interactúen con los servicios de AWS mediante los permisos definidos en la función.

Entre los casos de uso habituales de las funciones de ejecución de IAM se incluyen los siguientes:

- Acceso a los datos de prueba almacenados en los buckets de Amazon S3
- Trasladar los artefactos de prueba a los buckets de Amazon S3
- Recuperación de la configuración de la aplicación de AWS AppConfig
- Escribir registros y métricas de pruebas en Amazon CloudWatch
- Envío de resultados de pruebas o mensajes de estado a las colas de Amazon SQS
- Llamar a las funciones de AWS Lambda como parte de los flujos de trabajo de prueba

Requisitos de rol de IAM

Para utilizar una función de ejecución de IAM con Device Farm, su función debe cumplir los siguientes requisitos:

- Relación de confianza: se debe confiar en el director del servicio Device Farm para que asuma la función. La política de confianza debe incluirse `devicefarm.amazonaws.com` como entidad de confianza.
- Permisos: el rol debe tener los permisos necesarios para acceder a los recursos de AWS que requieren sus pruebas.
- Duración de la sesión: la duración máxima de la sesión del rol debe ser como mínimo igual a la configuración de tiempo de espera del trabajo del proyecto de Device Farm. De forma predeterminada, los proyectos de Device Farm tienen un tiempo de espera de trabajo de 150 minutos, por lo que su función debe admitir una duración de sesión de al menos 150 minutos.

- El mismo requisito de cuenta: la función de IAM debe estar en la misma cuenta de AWS que la utilizada para llamar a Device Farm. No se admite la suposición de roles entre cuentas.
- PassRole permiso: la persona que llama debe estar autorizada a pasar la función de IAM mediante una política que permita realizar `iam:PassRole` acciones en la función de ejecución especificada.

Política de confianza de ejemplo

El siguiente ejemplo muestra una política de confianza que permite a Device Farm asumir su función de ejecución. Esta política de confianza solo debe adjuntarse a la función de IAM específica que pretenda utilizar con Device Farm, no a otras funciones de su cuenta:

Example

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

Ejemplo de política de permisos

El siguiente ejemplo muestra una política de permisos que concede acceso a los servicios de AWS más comunes que se utilizan en las pruebas:

Example

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

Configuración de una función de ejecución de IAM

Puede especificar una función de ejecución de IAM a nivel de proyecto o para pruebas individuales. Cuando se configura a nivel de proyecto, todas las ejecuciones de ese proyecto heredarán la función de ejecución. Una función de ejecución configurada en una ejecución sustituirá a cualquier función configurada en su proyecto principal.

Para obtener instrucciones detalladas sobre la configuración de las funciones de ejecución, consulte:

- Creación de un proyecto en AWS Device Farm- para configurar las funciones de ejecución a nivel de proyecto
- Creación de una ejecución de prueba en Device Farm- para configurar las funciones de ejecución para ejecuciones individuales

También puede configurar las funciones de ejecución mediante la API Device Farm. Para obtener más información, consulte la referencia de la API de Device Farm.

Prácticas recomendadas

Siga estas prácticas recomendadas al configurar las funciones de ejecución de IAM para sus pruebas de Device Farm:

- Principio del mínimo privilegio: conceda solo los permisos mínimos necesarios para que las pruebas funcionen. Evita usar permisos demasiado amplios, como * acciones o recursos.
- Utilice permisos específicos para cada recurso: cuando sea posible, limite los permisos a recursos específicos (p. ej., cubos de S3 específicos o tablas de DynamoDB) en lugar de a todos los recursos de un mismo tipo.
- Recursos de prueba y de producción separados: utilice funciones y recursos de prueba específicos para evitar que los sistemas de producción se vean afectados accidentalmente durante las pruebas.
- Revisión periódica de las funciones: revise y actualice periódicamente sus funciones de ejecución para asegurarse de que siguen satisfaciendo sus necesidades de pruebas y siguiendo las mejores prácticas de seguridad.
- Use claves de condición: considere la posibilidad de usar claves de condición de IAM para restringir aún más cuándo y cómo se puede usar el rol.

Resolución de problemas

Si tiene problemas con las funciones de ejecución de IAM, compruebe lo siguiente:

- Relación de confianza: compruebe que la política de confianza del rol se incluya `devicefarm.amazonaws.com` como un servicio de confianza.
- Permisos: compruebe que el rol tiene los permisos necesarios para los servicios de AWS a los que están intentando acceder las pruebas.
- Registros de pruebas: revise los registros de ejecución de las pruebas para ver mensajes de error específicos relacionados con las llamadas a las API de AWS o las denegaciones de permisos.

Variables de entorno para entornos de prueba personalizados

Device Farm configura dinámicamente varias variables de entorno para usarlas como parte de la ejecución de un entorno de pruebas personalizado.

Temas

Variables de entorno personalizadas

Device Farm admite la configuración de pares clave-valor que se aplican como variables de entorno en el host de prueba. Se pueden configurar en un proyecto de Device Farm o durante la creación de una ejecución; cualquier variable configurada en una ejecución sustituirá a las que se hayan configurado en su proyecto principal. Se aplican las siguientes restricciones:

- Las variables de entorno personalizadas no se admiten en los hosts de prueba de iOS antiguos. Para obtener más información, consulte Host de pruebas de iOS antiguo.
- Los nombres de variables que comiencen por `$DEVICEFARM_` están reservados para el uso del servicio interno.
- Las variables de entorno personalizadas no se pueden usar para configurar la selección de cómputo del host de prueba en la especificación de la prueba.

Variables de entorno comunes

En esta sección se describen las variables de entorno comunes a todas las pruebas de Device Farm.

$DEVICEFARM_DEVICE_NAME

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

$DEVICEFARM_DEVICE_UDID

El identificador único del dispositivo.

$DEVICEFARM_DEVICE_PLATFORM_NAME

El nombre de la plataforma del dispositivo. Es `Android` o `iOS`.

$DEVICEFARM_DEVICE_OS_VERSION

La versión del sistema operativo del dispositivo.

$DEVICEFARM_APP_PATH

(pruebas de aplicaciones móviles)

La ruta a la aplicación móvil en la máquina host donde se ejecutan las pruebas. Esta variable no está disponible durante las pruebas web.

$DEVICEFARM_LOG_DIR

La ruta al directorio predeterminado donde se almacenarán los registros de los clientes, los artefactos y otros archivos necesarios para su posterior recuperación. Utilizando un ejemplo de especificación de prueba, 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_PROJECT_ARN

El ARN del proyecto principal del trabajo.

$DEVICEFARM_RUN_ARN

El ARN de la carrera principal del trabajo.

$DEVICEFARM_DEVICE_ARN

El ARN del dispositivo que se está probando.

$DEVICEFARM_TOTAL_JOBS

El número total de trabajos asociados a su ejecución principal de Device Farm.

$DEVICEFARM_JOB_NUMBER

El número de este trabajo está dentro de`$DEVICEFARM_TOTAL_JOBS`. Por ejemplo, una ejecución puede contener 5 trabajos y cada uno tendrá un `$DEVICEFARM_JOB_NUMBER` intervalo único de 0 a 4.

$AWS_REGION

La región de AWS. El servicio lo configurará para que coincida con la región en la que se encuentra el dispositivo que se está probando. Si es necesario, se puede anular mediante una variable de entorno personalizada.

$ANDROID_HOME

(Solo Android)

La ruta al directorio de instalación del SDK de Android.

Variables de entorno para las pruebas de Appium

En esta sección se describen las variables de entorno utilizadas por cualquier prueba de Appium en un entorno de pruebas personalizado en Device Farm.

$DEVICEFARM_CHROMEDRIVER_EXECUTABLE_DIR

(Solo Android)

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

$DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V<N>

(solo iOS)

La ruta de datos derivada de una versión de WebDriverAgent creada para ejecutarse en Device Farm. La numeración de la variable corresponderá a la versión principal de WebDriverAgent.

Como ejemplo, DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V9 apuntará a una WebDriverAgent versión de 9.x. Para obtener más información, consulte Selección de una WebDriverAgent versión para las pruebas de iOS.

> **Note**
>
> Las variables de $DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V<N> entorno solo están presentes en los hosts iOS que no son antiguos. Para obtener más información, consulte Host de pruebas de iOS antiguo.

$DEVICEFARM_WDA_DERIVED_DATA_PATH_V9

(solo para iOS, obsoleto)

La ruta de datos derivada de una versión de WebDriverAgent creada para ejecutarse en Device Farm. Consulte el esquema $DEVICEFARM_APPIUM_WDA_DERIVED_DATA_PATH_V<N> de nombres de reemplazo.

Variables de entorno para XCUITest las pruebas

En esta sección, se describen las variables de entorno que utiliza la XCUITest prueba en un entorno de prueba personalizado en Device Farm.

$DEVICEFARM_XCUITESTRUN_FILE

La ruta al `.xctestun` archivo 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.

$DEVICEFARM_XCTEST_BUILD_DIRECTORY

Ruta al contenido descomprimido del archivo del paquete de prueba.

Mejores prácticas para la ejecución de entornos de pruebas personalizados

En los temas siguientes se describen las prácticas recomendadas para utilizar la ejecución de pruebas personalizadas con Device Farm.

Ejecutar configuración

- Confíe en el software gestionado por Device Farm y en las funciones de la API para ejecutar la configuración siempre que sea posible, en lugar de aplicar configuraciones similares mediante comandos de shell en el archivo de especificaciones de prueba. Esto incluye la configuración del host de prueba y del dispositivo, ya que será más sostenible y coherente en todos los hosts y dispositivos de prueba.

  Si bien Device Farm lo alienta a personalizar el archivo de especificaciones de prueba tanto como necesite para ejecutar las pruebas, el archivo de especificaciones de prueba puede resultar difícil de mantener con el tiempo a medida que se le agreguen más comandos personalizados. Utilizar el software gestionado Device Farm (a través de herramientas como `devicefarm-cli` las herramientas predeterminadas disponibles en el`$PATH`) y utilizar funciones gestionadas (como el parámetro de `deviceProxy`solicitud) para simplificar el archivo de especificaciones de las pruebas al transferir la responsabilidad del mantenimiento al propio Device Farm.

Pruebe las especificaciones y el código del paquete

- No utilice rutas absolutas ni confíe en versiones secundarias específicas del archivo de especificaciones de prueba o del código del paquete de prueba. Device Farm aplica actualizaciones de rutina al host de prueba seleccionado y a las versiones de software incluidas. El uso de rutas específicas o absolutas (por ejemplo, `/usr/local/bin/python` en lugar de`python`) o la exigencia de versiones secundarias específicas (por ejemplo, Node.js `20.3.1` en lugar de solo `20`) puede provocar que las pruebas no localicen el archivo o ejecutable requerido.

  Como parte de la ejecución de las pruebas personalizadas, Device Farm configura varias variables de entorno y la `$PATH` variable para garantizar que las pruebas tengan una experiencia coherente en nuestros entornos dinámicos. Para obtener más información, consulte Variables de entorno para entornos de prueba personalizados y Software compatible en entornos de prueba personalizados.

- Guarde los archivos generados o copiados en el directorio temporal durante la ejecución de la prueba. Hoy nos aseguramos de que el usuario pueda acceder al directorio temporal (`/tmp`) durante la ejecución de la prueba (además de los directorios gestionados, como el`$DEVICEFARM_LOG_DIR`). Otros directorios a los que el usuario tiene acceso pueden cambiar con el tiempo debido a las necesidades del servicio o del sistema operativo utilizado.
- Guarde los registros de ejecución de las pruebas en **`$DEVICEFARM_LOG_DIR`**. Este es el directorio de artefactos predeterminado que se proporciona para que la ejecución añada registros de ejecución o artefactos. Cada uno de los ejemplos de especificaciones de prueba que proporcionamos utiliza este directorio para los artefactos de forma predeterminada.
- Asegúrese de que sus comandos devuelvan un código distinto de cero en caso de error durante la `test` fase de la especificación de prueba. Determinamos si la ejecución ha fallado comprobando si hay un código de salida distinto de cero en cada comando de shell invocado durante la fase. `test` Debe asegurarse de que su estructura lógica o de prueba devuelva un código de salida distinto de cero para todos los escenarios deseados, que pueden requerir una configuración adicional.

  Por ejemplo, algunos marcos de pruebas (por ejemplo JUnit5) no consideran que la ejecución de cero pruebas sea un error, lo que provocará que se detecte que las pruebas se han ejecutado correctamente aunque no se haya ejecutado nada. JUnit5 Como ejemplo, tendría que especificar la opción de línea de comandos `--fail-if-no-tests` para garantizar que este escenario se cierre con un código de salida distinto de cero.
- Revise la compatibilidad del software con la versión del sistema operativo del dispositivo y la versión del host que utilizará para la ejecución de la prueba. Por ejemplo, hay ciertas características de los marcos de software de prueba (por ejemplo, Appium) que pueden no funcionar según lo previsto en todas las versiones del sistema operativo del dispositivo que se está probando.

Seguridad

- Evite almacenar o registrar variables sensibles (como las claves de AWS) en el archivo de especificaciones de la prueba. Los archivos de especificaciones de prueba, los scripts generados por las especificaciones de prueba y los registros del script de especificaciones de prueba se proporcionan como elementos descargables al final de la ejecución de la prueba. Esto puede provocar que otros usuarios de tu cuenta que tengan acceso de lectura a la prueba revelen información secreta de forma no intencionada.

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

Puede cambiar de un modo de ejecución de pruebas estándar a un modo de ejecución personalizado en AWS Device Farm. 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 video en directo.

Para obtener más información sobre los modos estándar y personalizado de Device Farm, consulte Entornos de prueba de AWS Device Farm y Personalización del entorno de pruebas personalizado en AWS Device Farm..

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, videos y otros artefactos de resultados de forma individual para cada prueba. Sin embargo, esto aumenta de manera constante el tiempo total de ejecución de las end-to-end pruebas, ya que el procesamiento previo y posterior de las pruebas y los resultados se ven alterados por parte del servicio.

   Por el contrario, el modo de ejecución personalizado no analiza su paquete de pruebas; esto significa que no hay preprocesamiento y que el postprocesamiento de las pruebas o de los artefactos resultantes es mínimo. 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. Video en directo y registro: los modos de ejecución estándar y personalizado le proporcionan videos y registros para sus pruebas. Sin embargo, en el modo estándar, solo obtendrá el video 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 video y de los registros de sus pruebas desde el lado del cliente. Además, podrá descargar el video y otros artefactos al final de las pruebas.

> **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 Consola de administración de AWS 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 necesario realizar cambios para mover las XCUITest pruebas de iOS a un entorno de pruebas personalizado.

Ampliación de los entornos de prueba personalizados en 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. 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.

Para obtener más información acerca de los entornos de pruebas personalizados en Device Farm, consulte Personalización del entorno de pruebas personalizado en AWS Device Farm..

Temas

Configuración de un PIN de dispositivo al ejecutar pruebas en Device Farm

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.

Para ver otras maneras de ampliar el conjunto de pruebas y optimizar las pruebas, consulte Ampliación de los entornos de prueba personalizados en Device Farm.

Agilización de las pruebas basadas en Appium en Device Farm 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 dañar algunos conjuntos de pruebas XPath basados porque se ha cambiado 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.

Para ver otras maneras de ampliar el conjunto de pruebas y optimizar las pruebas, consulte Ampliación de los entornos de prueba personalizados en Device Farm.

Uso de Webhooks y otros APIs después de ejecutar las pruebas en Device Farm

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.

Para ver otras maneras de ampliar el conjunto de pruebas y optimizar las pruebas, consulte Ampliación de los entornos de prueba personalizados en Device Farm.

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 realizan solicitudes a otros APIs. 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 comprobar los requisitos de seguridad, consulta la documentación del tercero APIs que utilices.

Si planea usar los AWS servicios 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.

Para ver otras maneras de ampliar el conjunto de pruebas y optimizar las pruebas, consulte Ampliación de los entornos de prueba personalizados en Device Farm.

Adición de archivos adicionales a su paquete de pruebas en Device Farm

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.

Puedes añadir archivos a tu archivo de prueba antes de cargarlo 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. En el caso de los archivos IPA, especialmente cuando se utilizan con XCUITests ellos, le recomendamos que coloque los archivos adicionales en una ubicación ligeramente diferente debido a la forma en que se diseñan los AWS Device Farm paquetes de prueba de iOS. Al crear la prueba de iOS, el directorio de la aplicación de prueba se ubicará dentro de otro directorio denominado*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 XCUITest paquetes, agrega cualquier archivo adicional al directorio que termine *.app* dentro del *Payload* directorio. 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, la AWS Device Farm descomprimirá automáticamente antes de la prueba y dejará los archivos descomprimidos en la ubicación con la *$DEVICEFARM_TEST_PACKAGE_PATH* variable de entorno. (Esto significa que si agregas un archivo llamado *extra_file* a la raíz del archivo, como en el primer ejemplo, se ubicará allí *$DEVICEFARM_TEST_PACKAGE_PATH/ extra_file* durante la prueba).

Para usar un ejemplo más práctico, si eres un usuario de Appium TestNG que quiere incluir un *testng.xml* archivo con tu prueba, puedes incluirlo en tu 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 propio archivo del paquete cargado 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 los 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, encontrarás los archivos adicionales descomprimidos en el `/tmp` directorio (por ejemplo,). `/tmp/extra_file` En el caso de un archivo IPA, como se explicó anteriormente, los archivos adicionales estarían en una ubicación ligeramente diferente dentro de la carpeta que termina en`.app`, que está 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` (se puede hacer referencia como). `/tmp/Payload/*.app/extra_file`

Para ver otras maneras de ampliar el conjunto de pruebas y optimizar las pruebas, consulte Ampliación de los entornos de prueba personalizados en Device Farm.

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 video 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. Si es posible, utilice alternativas desarrolladas específicamente para las pruebas, como las cuentas de prueba.

Temas

Creación de 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 Creación de un proyecto en AWS Device Farm y, a continuación, vuelva a esta página.

Crear una sesión remota

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. Si ya tiene un proyecto, selecciónelo de la lista. De lo contrario, para crear un proyecto, siga las instrucciones de Creación de un proyecto en AWS Device Farm.
4. En la pestaña Acceso remoto, seleccione Crear sesión de acceso remoto.
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.
6. (Opcional) Incluye una aplicación y aplicaciones auxiliares como parte de la sesión. Pueden ser aplicaciones recién cargadas o aplicaciones que se hayan cargado anteriormente en este proyecto en los últimos 30 días (después de 30 días, las cargas de aplicaciones caducarán).
7. En Nombre de sesión, escriba un nombre para la sesión.
8. Seleccione Confirmar e iniciar sesión.

AWS CLI

Nota: estas instrucciones se centran únicamente en la creación de una sesión de acceso remoto. Para obtener instrucciones sobre cómo cargar una aplicación para usarla durante la sesión, consulta cómo automatizar la carga de aplicaciones.

En primer lugar, compruebe que su versión de AWS CLI esté up-to-date disponible descargando e instalando la versión más reciente.

> **Important**
>
> Algunos comandos que se mencionan en este documento no están disponibles en las versiones anteriores de la AWS CLI.

A continuación, puede determinar en qué dispositivo desea realizar la prueba:

```
$ aws devicefarm list-devices
```

Esto mostrará un resultado como el siguiente:

```
{
    "devices":
    [
        {
            "arn": "arn:aws:devicefarm:us-
west-2::device:DE5BD47FF3BD42C3A14BF7A6EFB1BFE7",
            "name": "Google Pixel 8",
            "remoteAccessEnabled": true,
            "availability": "HIGHLY_AVAILABLE"
            ...
        },
        ...
    ]
}
```

A continuación, puede crear su sesión de acceso remoto con el ARN del dispositivo que prefiera:

```
$ aws devicefarm create-remote-access-session \
  --project-arn arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef \
  --device-arn arn:aws:devicefarm:us-west-2::device:DE5BD47FF3BD42C3A14BF7A6EFB1BFE7
 \
  --app-arn arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
 \
  --configuration '{
      "auxiliaryApps": [
          "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
```

```
          "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
      ]
  }'
```

Esto mostrará un resultado como el siguiente:

```
{
    "remoteAccessSession": {
        "arn": "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000",
        "name": "Google Pixel 8",
        "status": "PENDING",
        ...
}
```

Ahora, opcionalmente, podemos sondear y esperar a que la sesión esté lista:

```
$ POLL_INTERVAL=3
TIMEOUT=600
DEADLINE=$(( $(date +%s) + TIMEOUT ))

while [[ "$(date +%s)" -lt "$DEADLINE" ]]; do

  STATUS=$(aws devicefarm get-remote-access-session \
    --arn "$DEVICE_FARM_SESSION_ARN" \
    --query 'remoteAccessSession.status' \
    --output text)

  case "$STATUS" in
    RUNNING)
      echo "Session is ready with status: $STATUS"
      break
      ;;
    STOPPING|COMPLETED)
      echo "Session ended early with status: $STATUS"
      exit 1
      ;;
  esac

done
```

Python

Nota: estas instrucciones se centran únicamente en la creación de una sesión de acceso remoto. Para obtener instrucciones sobre cómo cargar una aplicación para usarla durante la sesión, consulta cómo automatizar la carga de aplicaciones.

En este ejemplo, primero se busca cualquier dispositivo Google Pixel disponible en Device Farm, se crea una sesión de acceso remoto con él y se espera a que se ejecute la sesión.

```
import random
import time
import boto3

client = boto3.client("devicefarm", region_name="us-west-2")

# 1) Gather all matching devices via paginated ListDevices with filters
filters = [
    {"attribute": "MODEL",        "operator": "CONTAINS", "values": ["Pixel"]},
    {"attribute": "AVAILABILITY", "operator": "EQUALS",   "values": ["AVAILABLE"]},
]

matching_arns = []
next_token = None
while True:
    args = {"filters": filters}
    if next_token:
        args["nextToken"] = next_token
    page = client.list_devices(**args)
    for d in page.get("devices", []):
        matching_arns.append(d["arn"])
    next_token = page.get("nextToken")
    if not next_token:
        break

if not matching_arns:
    raise RuntimeError("No available Google Pixel device found.")

# Randomly select one device from the full matching set
device_arn = random.choice(matching_arns)
print("Selected device ARN:", device_arn)

# 2) Create remote access session and wait until RUNNING
resp = client.create_remote_access_session(
```

```
    projectArn="arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef",
    deviceArn=device_arn,
    appArn="arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
  # optional
    configuration={
        "auxiliaryApps": [  # optional
            "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
            "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
        ]
    },
)

session_arn = resp["remoteAccessSession"]["arn"]
print(f"Created Remote Access Session: {session_arn}")

poll_interval = 3
timeout = 600
deadline = time.time() + timeout
terminal_states = ["STOPPING", "COMPLETED"]

while True:
    out = client.get_remote_access_session(arn=session_arn)
    status = out["remoteAccessSession"]["status"]
    print(f"Current status: {status}")

    if status == "RUNNING":
        print(f"Session is ready with status: {status}")
        break
    if status in terminal_states:
        raise RuntimeError(f"Session ended early with status: {status}")
    if time.time() >= deadline:
        raise RuntimeError("Timed out waiting for session to be ready.")
    time.sleep(poll_interval)
```

Java

Nota: estas instrucciones se centran únicamente en la creación de una sesión de acceso remoto. Para obtener instrucciones sobre cómo cargar una aplicación para usarla durante la sesión, consulta cómo automatizar la carga de aplicaciones.

Nota: en este ejemplo se utiliza el AWS SDK for Java v2 y es compatible con las versiones 11 y superiores de JDK.

En este ejemplo, primero se busca cualquier dispositivo Google Pixel disponible en Device Farm, se crea una sesión de acceso remoto con él y se espera a que se ejecute la sesión.

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.concurrent.ThreadLocalRandom;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import
 software.amazon.awssdk.services.devicefarm.model.CreateRemoteAccessSessionConfiguration;
import
 software.amazon.awssdk.services.devicefarm.model.CreateRemoteAccessSessionRequest;
import
 software.amazon.awssdk.services.devicefarm.model.CreateRemoteAccessSessionResponse;
import software.amazon.awssdk.services.devicefarm.model.Device;
import software.amazon.awssdk.services.devicefarm.model.DeviceFilter;
import software.amazon.awssdk.services.devicefarm.model.DeviceFilterAttribute;
import
 software.amazon.awssdk.services.devicefarm.model.GetRemoteAccessSessionRequest;
import
 software.amazon.awssdk.services.devicefarm.model.GetRemoteAccessSessionResponse;
import software.amazon.awssdk.services.devicefarm.model.ListDevicesRequest;
import software.amazon.awssdk.services.devicefarm.model.ListDevicesResponse;
import software.amazon.awssdk.services.devicefarm.model.RuleOperator;

public class CreateRemoteAccessSession {
  public static void main(String[] args) throws Exception {
    DeviceFarmClient client = DeviceFarmClient.builder()
        .region(Region.US_WEST_2)
        .build();

    String projectArn = "arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef";
    String appArn     = "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
    String aux1       = "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
    String aux2       = "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
```

```
    // 1) Gather all matching devices via paginated ListDevices with filters
    List<DeviceFilter> filters = Arrays.asList(
        DeviceFilter.builder()
            .attribute(DeviceFilterAttribute.MODEL)
            .operator(RuleOperator.CONTAINS)
            .values("Pixel")
            .build(),
        DeviceFilter.builder()
            .attribute(DeviceFilterAttribute.AVAILABILITY)
            .operator(RuleOperator.EQUALS)
            .values("AVAILABLE")
            .build()
    );

    List<String> matchingDeviceArns = new ArrayList<>();
    String next = null;
    do {
      ListDevicesResponse page = client.listDevices(
          ListDevicesRequest.builder().filters(filters).nextToken(next).build());
      for (Device d : page.devices()) {
        matchingDeviceArns.add(d.arn());
      }
      next = page.nextToken();
    } while (next != null);

    if (matchingDeviceArns.isEmpty()) {
      throw new RuntimeException("No available Google Pixel device found.");
    }

    // Randomly select one device from the full matching set
    String deviceArn = matchingDeviceArns.get(
        ThreadLocalRandom.current().nextInt(matchingDeviceArns.size()));
    System.out.println("Selected device ARN: " + deviceArn);

    // 2) Create Remote Access session and wait until it is RUNNING
    CreateRemoteAccessSessionConfiguration cfg =
CreateRemoteAccessSessionConfiguration.builder()
        .auxiliaryApps(Arrays.asList(aux1, aux2))
        .build();

    CreateRemoteAccessSessionResponse res = client.createRemoteAccessSession(
        CreateRemoteAccessSessionRequest.builder()
            .projectArn(projectArn)
```

```
            .deviceArn(deviceArn)
            .appArn(appArn)       // optional
            .configuration(cfg)   // optional
            .build());

    String sessionArn = res.remoteAccessSession().arn();
    System.out.println("Created Remote Access Session: " + sessionArn);

    int pollIntervalMs = 3000;
    long timeoutMs = 600_000L;
    long deadline = System.currentTimeMillis() + timeoutMs;

    while (true) {
      GetRemoteAccessSessionResponse get = client.getRemoteAccessSession(
          GetRemoteAccessSessionRequest.builder().arn(sessionArn).build());
      String status = get.remoteAccessSession().statusAsString();
      System.out.println("Current status: " + status);

      if ("RUNNING".equals(status)) {
        System.out.println("Session is ready with status: " + status);
        break;
      }
      if ("STOPPING".equals(status) || "COMPLETED".equals(status)) {
        throw new RuntimeException("Session ended early with status: " + status);
      }
      if (System.currentTimeMillis() >= deadline) {
        throw new RuntimeException("Timed out waiting for session to be ready.");
      }
      Thread.sleep(pollIntervalMs);
    }
  }
}
```

JavaScript

Nota: estas instrucciones se centran únicamente en la creación de una sesión de acceso remoto. Para obtener instrucciones sobre cómo cargar una aplicación para usarla durante la sesión, consulta cómo automatizar la carga de aplicaciones.

Nota: en este ejemplo se utiliza AWS SDK para la JavaScript versión 3.

En este ejemplo, primero se busca cualquier dispositivo Google Pixel disponible en Device Farm, se crea una sesión de acceso remoto con él y se espera a que se ejecute la sesión.

```
import {
  DeviceFarmClient,
  ListDevicesCommand,
  CreateRemoteAccessSessionCommand,
  GetRemoteAccessSessionCommand,
} from "@aws-sdk/client-device-farm";

const client = new DeviceFarmClient({ region: "us-west-2" });

// 1) Gather all matching devices via paginated ListDevices with filters
const filters = [
  { attribute: "MODEL",        operator: "CONTAINS", values: ["Pixel"] },
  { attribute: "AVAILABILITY", operator: "EQUALS",   values: ["AVAILABLE"] },
];

let nextToken;
const matching = [];

while (true) {
  const page = await client.send(new ListDevicesCommand({ filters, nextToken }));
  for (const d of page.devices ?? []) {
    matching.push(d.arn);
  }
  nextToken = page.nextToken;
  if (!nextToken) break;
}

if (matching.length === 0) {
  throw new Error("No available Google Pixel device found.");
}

// Randomly select one device from the full matching set
const deviceArn = matching[Math.floor(Math.random() * matching.length)];
console.log("Selected device ARN:", deviceArn);

// 2) Create remote access session and wait until RUNNING
const out = await client.send(new CreateRemoteAccessSessionCommand({
  projectArn: "arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef",
  deviceArn,
  appArn: "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
 optional
```

```
  configuration: {
    auxiliaryApps: [ // optional
      "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
      "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
    ],
  },
}));

const sessionArn = out.remoteAccessSession?.arn;
console.log("Created Remote Access Session:", sessionArn);

const pollIntervalMs = 3000;
const timeoutMs = 600000;
const deadline = Date.now() + timeoutMs;

while (true) {
  const get = await client.send(new GetRemoteAccessSessionCommand({ arn:
 sessionArn }));
  const status = get.remoteAccessSession?.status;
  console.log("Current status:", status);

  if (status === "RUNNING") {
    console.log("Session is ready with status:", status);
    break;
  }
  if (status === "STOPPING" || status === "COMPLETED") {
    throw new Error(`Session ended early with status: ${status}`);
  }
  if (Date.now() >= deadline) {
    throw new Error("Timed out waiting for session to be ready.");
  }
  await new Promise((r) => setTimeout(r, pollIntervalMs));
}
```

C#

Nota: estas instrucciones se centran únicamente en la creación de una sesión de acceso remoto. Para obtener instrucciones sobre cómo cargar una aplicación para usarla durante la sesión, consulta cómo automatizar la carga de aplicaciones.

En este ejemplo, primero se busca cualquier dispositivo Google Pixel disponible en Device Farm, se crea una sesión de acceso remoto con él y se espera a que se ejecute la sesión.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

class Program
{
    static async Task Main()
    {
        var client = new AmazonDeviceFarmClient(RegionEndpoint.USWest2);

        // 1) Gather all matching devices via paginated ListDevices with filters
        var filters = new List<DeviceFilter>
        {
            new DeviceFilter { Attribute = DeviceAttribute.MODEL,        Operator =
 RuleOperator.CONTAINS, Values = new List<string>{ "Pixel" } },
            new DeviceFilter { Attribute = DeviceAttribute.AVAILABILITY, Operator =
 RuleOperator.EQUALS,   Values = new List<string>{ "AVAILABLE" } },
        };

        var matchingArns = new List<string>();
        string nextToken = null;

        do
        {
            var list = await client.ListDevicesAsync(new ListDevicesRequest
            {
                Filters = filters,
                NextToken = nextToken
            });

            foreach (var d in list.Devices)
                matchingArns.Add(d.Arn);

            nextToken = list.NextToken;
        }
        while (nextToken != null);
```

```
        if (matchingArns.Count == 0)
            throw new Exception("No available Google Pixel device found.");

        // Randomly select one device from the full matching set
        var rnd = new Random();
        var deviceArn = matchingArns[rnd.Next(matchingArns.Count)];
        Console.WriteLine($"Selected device ARN: {deviceArn}");

        // 2) Create remote access session and wait until RUNNING
        var request = new CreateRemoteAccessSessionRequest
        {
            ProjectArn = "arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef",
            DeviceArn  = deviceArn,
            AppArn     = "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
 optional
            Configuration = new CreateRemoteAccessSessionConfiguration
            {
                AuxiliaryApps = new List<string>
                {
                    "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
                    "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
                }
            }
        };

        request.Configuration.AuxiliaryApps.RemoveAll(string.IsNullOrWhiteSpace);

        var response = await client.CreateRemoteAccessSessionAsync(request);
        var sessionArn = response.RemoteAccessSession.Arn;
        Console.WriteLine($"Created Remote Access Session: {sessionArn}");

        var pollIntervalMs = 3000;
        var timeoutMs = 600000;
        var deadline = DateTime.UtcNow.AddMilliseconds(timeoutMs);

        while (true)
        {
            var get = await client.GetRemoteAccessSessionAsync(new
 GetRemoteAccessSessionRequest { Arn = sessionArn });
            var status = get.RemoteAccessSession.Status.Value;
```

```
            Console.WriteLine($"Current status: {status}");

            if (status == "RUNNING")
            {
                Console.WriteLine($"Session is ready with status: {status}");
                break;
            }
            if (status == "STOPPING" || status == "COMPLETED")
            {
                throw new Exception($"Session ended early with status: {status}");
            }
            if (DateTime.UtcNow >= deadline)
            {
                throw new TimeoutException("Timed out waiting for session to be
 ready.");
            }

            await Task.Delay(pollIntervalMs);
        }
    }
}
```

Ruby

Nota: estas instrucciones se centran únicamente en la creación de una sesión de acceso remoto. Para obtener instrucciones sobre cómo cargar una aplicación para usarla durante la sesión, consulta cómo automatizar la carga de aplicaciones.

En este ejemplo, primero se busca cualquier dispositivo Google Pixel disponible en Device Farm, se crea una sesión de acceso remoto con él y se espera a que se ejecute la sesión.

```
require 'aws-sdk-devicefarm'

client = Aws::DeviceFarm::Client.new(region: 'us-west-2')

# 1) Gather all matching devices via paginated ListDevices with filters
filters = [
  { attribute: 'MODEL',        operator: 'CONTAINS', values: ['Pixel'] },
  { attribute: 'AVAILABILITY', operator: 'EQUALS',   values: ['AVAILABLE'] },
]

matching_arns = []
next_token = nil
```

```
loop do
  resp = client.list_devices(filters: filters, next_token: next_token)
  resp.devices&.each { |d| matching_arns << d.arn }
  next_token = resp.next_token
  break unless next_token
end

abort "No available Google Pixel device found." if matching_arns.empty?

# Randomly select one device from the full matching set
device_arn = matching_arns.sample
puts "Selected device ARN: #{device_arn}"

# 2) Create remote access session and wait until RUNNING
resp = client.create_remote_access_session(
  project_arn: "arn:aws:devicefarm:us-
west-2:111122223333:project:12345678-1111-2222-333-456789abcdef",
  device_arn:  device_arn,
  app_arn:     "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
 # optional
  configuration: {
    auxiliary_apps: [ # optional
      "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
      "arn:aws:devicefarm:us-
west-2:111122223333:upload:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-456789
    ].compact
  }
)

session_arn = resp.remote_access_session.arn
puts "Created Remote Access Session: #{session_arn}"

poll_interval = 3
timeout = 600
deadline = Time.now + timeout
terminal = %w[STOPPING COMPLETED]

loop do
  get = client.get_remote_access_session(arn: session_arn)
  status = get.remote_access_session.status
  puts "Current status: #{status}"
```

```
  if status == 'RUNNING'
    puts "Session is ready with status: #{status}"
    break
  end

  abort "Session ended early with status: #{status}" if terminal.include?(status)
  abort "Timed out waiting for session to be ready." if Time.now >= deadline
  sleep poll_interval
end
```

Siguientes pasos

Device Farm inicia la sesión en cuanto el dispositivo y la infraestructura solicitados estén disponibles, normalmente en unos minutos. Se muestra el cuadro de diálogo Dispositivo solicitado hasta que comienza la sesión. Para cancelar la solicitud de sesión, seleccione Cancelar solicitud.

Si el dispositivo seleccionado no está disponible o está ocupado, el estado de la sesión se muestra como Dispositivo pendiente, lo que indica que es posible que deba esperar algún tiempo antes de que el dispositivo esté disponible para realizar pruebas.

Si tu cuenta ha alcanzado el límite de simultaneidad para dispositivos públicos con o sin contador, el estado de la sesión se muestra como pendiente de simultaneidad. En el caso de las ranuras para dispositivos sin contador, puedes aumentar la simultaneidad comprando más ranuras para dispositivos. En el caso de pay-as-you-go los dispositivos con contador, póngase en contacto con AWS a través de un ticket de soporte para solicitar un aumento de la cuota de servicio.

Cuando se inicia la configuración de la sesión, primero se muestra el estado En curso y, a continuación, el estado de Conexión, mientras el navegador web local intenta abrir una conexión remota con el dispositivo.

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 iniciada la sesión, puedes interactuar con el dispositivo en el navegador web o probar el dispositivo con Appium.

Uso de 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

- Crear una sesión. Siga las instrucciones de Creación de 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**
>
> Cuando carga una aplicación, a veces existe un retraso antes de que la aplicación esté disponible. Aparecerá un mensaje de confirmación para informarte si la aplicación se instaló correctamente o no.

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.

Siguientes pasos

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 video capturado y a los registros generados. Para obtener más información, consulte Recuperación de los resultados de la sesión.

Sugerencias y trucos

Es posible que experimente problemas de rendimiento con la sesión de acceso remoto en algunas AWS regiones. 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.

Obtención de 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 video de una sesión

Requisitos previos

- Complete una sesión. Siga las instrucciones de Uso de 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 video de una sesión

Cuando una sesión de acceso remoto finaliza, la consola de Device Farm proporciona acceso a una captura de video 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 video de la sesión. Puede ver estos archivos en el navegador o guardarlos localmente.

Pruebas de Appium en AWS Device Farm

Durante una sesión de acceso remoto, puede ejecutar pruebas de Appium desde su entorno local y dirigirlas al dispositivo de la sesión mediante un punto de conexión de Appium administrado. Con un punto final de Appium, puedes desarrollar, probar y ejecutar el código de Appium con una respuesta rápida y una iteración rápidas. Este enfoque de pruebas del lado del cliente ofrece la flexibilidad de conectarse a un dispositivo Device Farm desde cualquier entorno de cliente de Appium que elija.

Para complementar las pruebas del lado del cliente, Device Farm también admite la ejecución de pruebas en la infraestructura administrada por el servicio, lo que se denomina ejecución del lado del servidor. Con este enfoque, puedes cargar la aplicación y las pruebas en el servicio y, a continuación, ejecutar las pruebas en paralelo en varios dispositivos mediante hosts de prueba gestionados por el servicio. Este enfoque se adapta bien para realizar pruebas en muchos dispositivos de forma independiente, así como para realizar pruebas desde el contexto de una canalización. CI/CD

Para obtener más información sobre la ejecución en el servidor, consulta. Marcos de pruebas y pruebas integradas

Temas

¿Qué es un punto final de Appium?

Appium es un popular marco de pruebas de software de código abierto para probar aplicaciones web nativas, híbridas y móviles en diferentes dispositivos, incluidos teléfonos móviles y tabletas, tanto para iOS como para Android. Permite a los desarrolladores e ingenieros de control de calidad (control de calidad) escribir scripts que pueden controlar un dispositivo de forma remota, simular las interacciones de los usuarios y verificar que la aplicación objeto de prueba se comporta como se espera. Appium interactúa con las aplicaciones desde la perspectiva del usuario final, lo que permite a los evaluadores desarrollar pruebas que simulan la forma en que los usuarios reales utilizarán la aplicación para sus pruebas.

Appium se basa en el modelo cliente-servidor, en el que un cliente local solicita a un servidor Appium (local o remoto) que controle un dispositivo en su nombre. El servidor Appium administra un controlador para comunicarse con el dispositivo, como el UIAutomator2 controlador para Android o el XCUITest controlador para iOS. Todos los comandos siguen los WebDriver estándares del W3C sobre cómo controlar un dispositivo.

El terminal Appium de Device Farm muestra la URL del servidor Appium del dispositivo en tu sesión de acceso remoto. La URL del punto de conexión de Appium será específica de ese dispositivo en esa sesión y seguirá siendo válida durante toda la sesión, lo que le permitirá realizar iteraciones en el mismo dispositivo sin necesidad de tiempo de configuración adicional. Para obtener más información sobre el acceso remoto, consulte. Acceso remoto

Cómo empezar con las pruebas de Appium

Para la mayoría de los usuarios de Appium, el uso de Device Farm para las pruebas de Appium solo requiere cambios menores en la configuración de prueba existente.

En términos generales, hay tres pasos para usar Device Farm para las pruebas de Appium del lado del cliente:

1. En primer lugar, debe crear una sesión de acceso remoto para probar un dispositivo Device Farm. Puedes incluir tus aplicaciones como parte de tu solicitud de acceso remoto o instalarlas una vez iniciada la sesión.
2. Una vez que se ejecute la sesión, puede copiar la URL del punto final de Appium y utilizarla a través de una herramienta independiente (como el Inspector de Appium) o desde el código de prueba de Appium en su IDE. La URL será válida durante toda la sesión de acceso remoto.
3. Por último, una vez que la prueba de Appium haya comenzado, podrás revisar los registros del servidor de Appium en tiempo real durante la ejecución de la prueba junto con la transmisión de vídeo de tu dispositivo.

Interactuar con el dispositivo mediante Appium

Una vez que haya creado una sesión de acceso remoto, el dispositivo estará disponible para las pruebas de Appium. Durante toda la sesión de acceso remoto, puedes ejecutar tantas sesiones de Appium como desees en el dispositivo, sin límites en cuanto a los clientes que utilices. Por ejemplo, puede empezar por ejecutar una prueba con el código Appium local de su IDE y, a continuación, pasar a utilizar el Inspector de Appium para solucionar cualquier problema que encuentre. La sesión

puede durar hasta 150 minutos; sin embargo, si no hay actividad durante más de 5 minutos (ya sea a través de la consola interactiva o del terminal de Appium), se agotará el tiempo de espera de la sesión.

Uso de aplicaciones para realizar pruebas con tu sesión de Appium

Device Farm te permite usar tus aplicaciones como parte de tu solicitud de creación de sesión de acceso remoto o instalar aplicaciones durante la propia sesión de acceso remoto. Estas aplicaciones se instalan automáticamente en el dispositivo que se está probando y se incorporan como funciones predeterminadas para cualquier solicitud de sesión de Appium. Al crear una sesión de acceso remoto, tiene la opción de transferir un ARN de aplicación, que se utilizará de forma predeterminada como `appium:app` capacidad para todas las sesiones de Appium posteriores, así como una aplicación auxiliar ARNs, que se utilizará como capacidad. `appium:otherApps`

Por ejemplo, si crea una sesión de acceso remoto con una aplicación `com.aws.devicefarm.sample` como aplicación y `com.aws.devicefarm.other.sample` como una de sus aplicaciones auxiliares, cuando vaya a crear una sesión de Appium, tendrá capacidades similares a las siguientes:

```
{
    "value":
    {
        "sessionId": "abcdef123456-1234-5678-abcd-abcdef123456",
        "capabilities":
        {
            "app": "/tmp/com.aws.devicefarm.sample.apk",
            "otherApps": "[\"/tmp/com.aws.devicefarm.other.sample.apk\"]",
            ...
        }
    }
}
```

Durante la sesión, puedes instalar aplicaciones adicionales (ya sea dentro de la consola o mediante la `InstallToRemoteAccessSession`API). Estas anularán cualquier aplicación existente que se haya utilizado anteriormente como `appium:app` capacidad. Si esas aplicaciones utilizadas anteriormente tienen un nombre de paquete distinto, permanecerán en el dispositivo y se utilizarán como parte de la `appium:otherApps` capacidad.

Por ejemplo, si inicialmente usas una aplicación `com.aws.devicefarm.sample` al crear tu sesión de acceso remoto, pero luego instalas una nueva aplicación con el nombre

`com.aws.devicefarm.other.sample` durante la sesión, tus sesiones de Appium tendrán capacidades similares a las siguientes:

```
{
    "value":
    {
        "sessionId": "abcdef123456-1234-5678-abcd-abcdef123456",
        "capabilities":
        {
            "app": "/tmp/com.aws.devicefarm.other.sample.apk",
            "otherApps": "[\"/tmp/com.aws.devicefarm.sample.apk\"]",
            ...
        }
    }
}
```

Si lo prefieres, puedes especificar de forma explícita las capacidades de tu aplicación mediante el nombre de la aplicación (utilizando las `appium:bundleId` capacidades `appium:appPackage` o las capacidades para Android e iOS, respectivamente).

Si estás probando una aplicación web, especifica la `browserName` capacidad de tu solicitud de creación de sesión de Appium. El `Chrome` navegador está disponible en todos los dispositivos Android y en todos los dispositivos iOS. `Safari`

Device Farm no admite el paso de una URL remota o una ruta del sistema de archivos local `appium:app` durante una sesión de acceso remoto. Cargue aplicaciones a Device Farm e inclúyalas en la sesión.

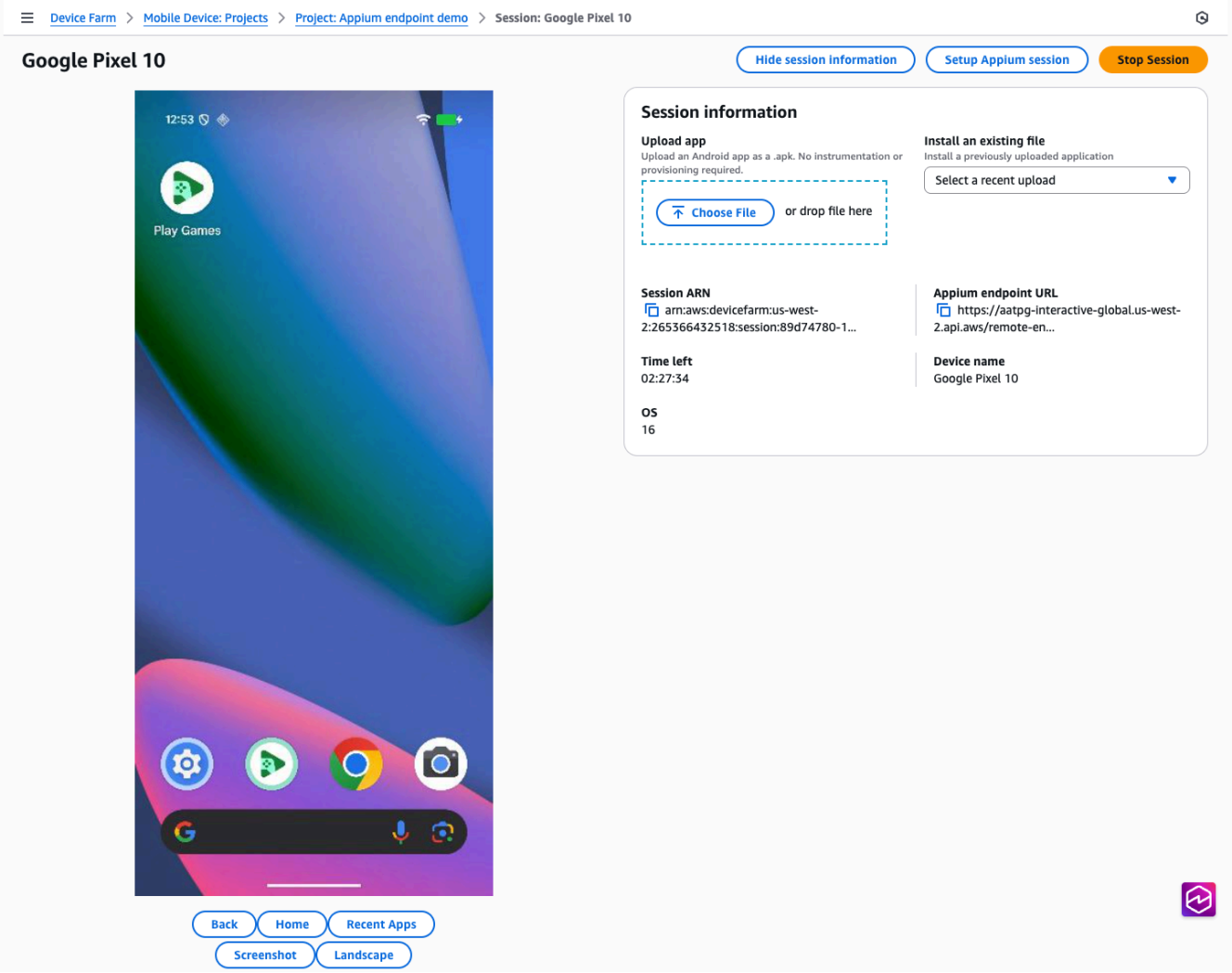
> **Note**
>
> Para obtener más información sobre la carga automática de aplicaciones como parte de tu sesión de acceso remoto, consulta cómo automatizar la carga de aplicaciones.

¿Cómo usar el terminal de Appium?

Estos son los pasos para acceder al punto final de Appium de la sesión desde la consola, el y el AWS CLI. AWS SDKs Estos pasos incluyen cómo empezar a ejecutar pruebas utilizando varios marcos de pruebas de clientes de Appium:

Console

1. Abre la página de sesión de acceso remoto en tu navegador web:



2. Para ejecutar una sesión con Appium Inspector, haga lo siguiente:

   a. Haga clic en el botón Configurar sesión de Appium

   b. Siga las instrucciones de la página sobre cómo iniciar una sesión con Appium Inspector.

3. Para ejecutar una prueba de Appium desde tu IDE local, haz lo siguiente:

   a. Haga clic en el icono de «copiar» situado junto al texto de la URL del punto de conexión de Appium

   b. Pegue esta URL en su código local de Appium donde especifique actualmente su dirección remota o ejecutor de comandos. Para ver ejemplos de idiomas específicos,

haga clic en una de las pestañas de esta ventana de ejemplo para el idioma que prefiera.

AWS CLI

En primer lugar, compruebe que su versión de AWS CLI esté up-to-date disponible descargando e instalando la versión más reciente.

> **Important**
>
> El campo de punto final de Appium no está disponible en las versiones anteriores de la AWS CLI.

Una vez que la sesión esté activa, la URL del punto de conexión de Appium estará disponible en un campo cuyo nombre aparece `remoteDriverEndpoint` en la respuesta a una llamada a la API: `GetRemoteAccessSession`

```
$ aws devicefarm get-remote-access-session \
    --arn "arn:aws:devicefarm:us-west-2:123456789876:session:abcdef123456-1234-5678-abcd-abcdef123456/abcdef123456-1234-5678-abcd-abcdef123456/00000"
```

Esto mostrará un resultado como el siguiente:

```
{
    "remoteAccessSession": {
        "arn": "arn:aws:devicefarm:us-west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/abcdef123456-1234-5678-abcd-abcdef123456/00000",
        "name": "Google Pixel 8",
        "status": "RUNNING",
        "endpoints": {
            "remoteDriverEndpoint": "https://devicefarm-interactive-global.us-west-2.api.aws/remote-endpoint/ABCD1234...",
        ...
}
```

Puedes usar esta URL en tu código local de Appium siempre que especifiques actualmente tu dirección remota o tu ejecutor de comandos. Para ver ejemplos de idiomas específicos, haga clic en una de las pestañas de esta ventana de ejemplo para seleccionar el idioma que prefiera.

Para ver un ejemplo de cómo interactuar con el punto final directamente desde la línea de comandos, puedes usar la herramienta de línea de comandos curl para llamar directamente al punto final: WebDriver

```
$ curl "https://devicefarm-interactive-global.us-west-2.api.aws/remote-endpoint/
ABCD1234.../status"
```

Esto mostrará un resultado como el siguiente:

```
{
    "value":
    {
        "ready": true,
        "message": "The server is ready to accept new connections",
        "build":
        {
            "version": "2.5.1"
        }
    }
}
```

Python

Una vez que la sesión esté en marcha, la URL del punto final de Appium estará disponible en un campo denominado `remoteDriverEndpoint` en la respuesta a una llamada a la `GetRemoteAccessSession`API:

```
# To get the URL
import sys
import boto3
from botocore.exceptions import ClientError

def get_appium_endpoint() -> str:
    session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000"
    device_farm_client = boto3.client("devicefarm", region_name="us-west-2")

    try:
        resp = device_farm_client.get_remote_access_session(arn=session_arn)
    except ClientError as exc:
        sys.exit(f"Failed to call Device Farm: {exc}")
```

```
    remote_access_session = resp.get("remoteAccessSession", {})
    endpoints = remote_access_session.get("endpoints", {})
    endpoint = endpoints.get("remoteDriverEndpoint")

    if not endpoint:
        sys.exit("Device Farm response did not include
 endpoints.remoteDriverEndpoint")

    return endpoint

# To use the URL
from appium import webdriver
from appium.options.android import UiAutomator2Options

opts = UiAutomator2Options()
driver = webdriver.Remote(get_appium_endpoint(), options=opts)
# ...
driver.quit()
```

Java

Nota: este ejemplo usa el AWS SDK para Java v2 y es compatible con las versiones 11 y superiores de JDK.

Una vez que la sesión esté en marcha, la URL del punto final de Appium estará disponible en un campo denominado `remoteDriverEndpoint` en la respuesta a una llamada a la API: GetRemoteAccessSession

```
// To get the URL
import software.amazon.awssdk.auth.credentials.DefaultCredentialsProvider;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import
 software.amazon.awssdk.services.devicefarm.model.GetRemoteAccessSessionRequest;
import
 software.amazon.awssdk.services.devicefarm.model.GetRemoteAccessSessionResponse;

public class AppiumEndpointBuilder {
    public static String getAppiumEndpoint() throws Exception {
        String session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000";
```

```
        try (DeviceFarmClient client = DeviceFarmClient.builder()
                .region(Region.US_WEST_2)
                .credentialsProvider(DefaultCredentialsProvider.create())
                .build()) {

            GetRemoteAccessSessionResponse resp = client.getRemoteAccessSession(
                    GetRemoteAccessSessionRequest.builder().arn(session_arn).build()
            );

            String endpoint =
 resp.remoteAccessSession().endpoints().remoteDriverEndpoint();
            if (endpoint == null || endpoint.isEmpty()) {
                throw new IllegalStateException("remoteDriverEndpoint missing from
 response");
            }
            return endpoint;
        }
    }
}

// To use the URL
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.android.options.UiAutomator2Options;

import java.net.URL;

public class ExampleTest {
    public static void main(String[] args) throws Exception {
        String endpoint = AppiumEndpointBuilder.getAppiumEndpoint();
        UiAutomator2Options options = new UiAutomator2Options();
        AndroidDriver driver = new AndroidDriver(new URL(endpoint), options);

        try {
            // ... your test ...
        } finally {
            driver.quit();
        }
    }
}
```

JavaScript

Nota: en este ejemplo se usa el AWS SDK para la versión JavaScript 3 y WebDriverIO para la versión 8+ con Node 18+.

Una vez que la sesión esté en marcha, la URL del punto final de Appium estará disponible en un campo denominado `remoteDriverEndpoint` en la respuesta a una llamada a la API: GetRemoteAccessSession

```
// To get the URL
import { DeviceFarmClient, GetRemoteAccessSessionCommand } from "@aws-sdk/client-
device-farm";

export async function getAppiumEndpoint() {
  const sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000";

  const client = new DeviceFarmClient({ region: "us-west-2" });
  const resp = await client.send(new GetRemoteAccessSessionCommand({ arn:
 sessionArn }));

  const endpoint = resp?.remoteAccessSession?.endpoints?.remoteDriverEndpoint;
  if (!endpoint) throw new Error("remoteDriverEndpoint missing from response");
  return endpoint;
}

// To use the URL with WebdriverIO
import { remote } from "webdriverio";

(async () => {
  const endpoint = await getAppiumEndpoint();
  const u = new URL(endpoint);

  const driver = await remote({
    protocol: u.protocol.replace(":", ""),
    hostname: u.hostname,
    port: u.port ? Number(u.port) : (u.protocol === "https:" ? 443 : 80),
    path: u.pathname + u.search,
    capabilities: {
      platformName: "Android",
      "appium:automationName": "UiAutomator2",
      // ...other caps...
```

```
    },
  });

  try {
    // ... your test ...
  } finally {
    await driver.deleteSession();
  }
})();
```

C#

Una vez que la sesión esté activa, la URL del punto de conexión de Appium estará disponible en un campo denominado `remoteDriverEndpoint` en la respuesta a una llamada a la API: GetRemoteAccessSession

```
// To get the URL
using System;
using System.Threading.Tasks;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

public static class AppiumEndpointBuilder
{
    public static async Task<string> GetAppiumEndpointAsync()
    {
        var sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000";

        var config = new AmazonDeviceFarmConfig
        {
            RegionEndpoint = RegionEndpoint.USWest2
        };
        using var client = new AmazonDeviceFarmClient(config);

        var resp = await client.GetRemoteAccessSessionAsync(new
 GetRemoteAccessSessionRequest { Arn = sessionArn });
        var endpoint = resp?.RemoteAccessSession?.Endpoints?.RemoteDriverEndpoint;

        if (string.IsNullOrWhiteSpace(endpoint))
```

```
            throw new InvalidOperationException("RemoteDriverEndpoint missing from
 response");

        return endpoint;
    }
}

// To use the URL
using OpenQA.Selenium.Appium;
using OpenQA.Selenium.Appium.Android;

class Example
{
    static async Task Main()
    {
        var endpoint = await AppiumEndpointBuilder.GetAppiumEndpointAsync();

        var options = new AppiumOptions();
        options.PlatformName = "Android";
        options.AutomationName = "UiAutomator2";

        using var driver = new AndroidDriver(new Uri(endpoint), options);
        try
        {
            // ... your test ...
        }
        finally
        {
            driver.Quit();
        }
    }
}
```

Ruby

Una vez que la sesión esté activa, la URL del punto de conexión de Appium estará disponible en un campo denominado `remoteDriverEndpoint` en la respuesta a una llamada a la API: `GetRemoteAccessSession`

```
# To get the URL
require 'aws-sdk-devicefarm'

def get_appium_endpoint
```

```
  session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:abcdef123456-1234-5678-abcd-abcdef123456/
abcdef123456-1234-5678-abcd-abcdef123456/00000"

  client = Aws::DeviceFarm::Client.new(region: 'us-west-2')
  resp = client.get_remote_access_session(arn: session_arn)
  endpoint = resp.remote_access_session.endpoints.remote_driver_endpoint
  raise "remote_driver_endpoint missing from response" if endpoint.nil? ||
 endpoint.empty?
  endpoint
end

# To use the URL
require 'appium_lib_core'

endpoint = get_appium_endpoint
opts = {
  server_url: endpoint,
  capabilities: {
    'platformName' => 'Android',
    'appium:automationName' => 'UiAutomator2'
  }
}

driver = Appium::Core.for(opts).start_driver
begin
  # ... your test ...
ensure
  driver.quit
end
```

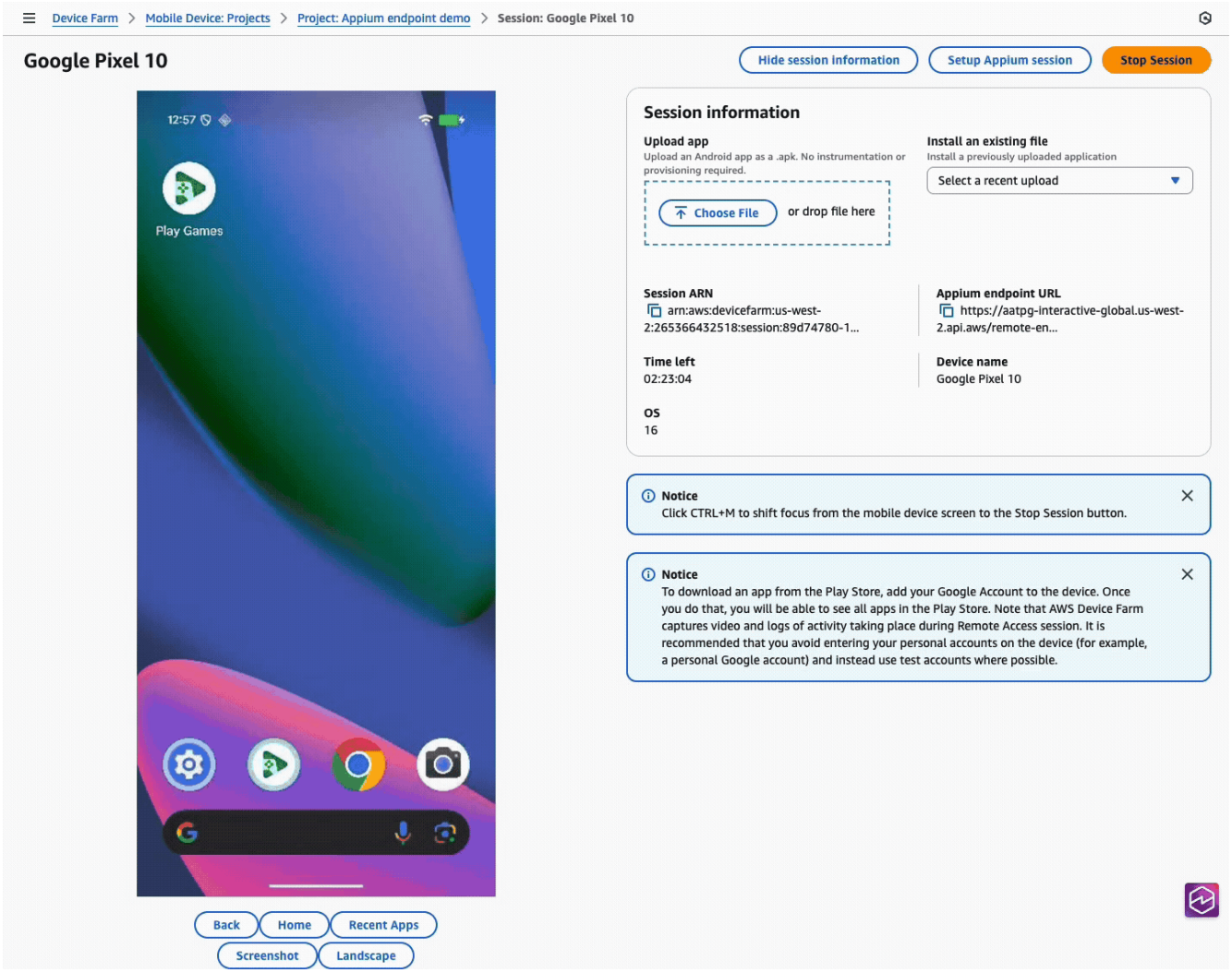
Revisión de los registros del servidor de Appium

Una vez que haya iniciado una sesión de Appium, podrá ver los registros del servidor de Appium en directo en la consola Device Farm o descargarlos cuando finalice la sesión de acceso remoto. Estas son las instrucciones para hacerlo:

Console

1. En la consola de Device Farm, abra la sesión de acceso remoto del dispositivo.

2. Inicie una sesión de punto final de Appium con el dispositivo desde su IDE local o el Inspector de Appium
3. Luego, el registro del servidor de Appium aparecerá junto al dispositivo en la página de la sesión de acceso remoto, con la «información de la sesión» disponible en la parte inferior de la página, debajo del dispositivo:



AWS CLI

Nota: en este ejemplo se utiliza la herramienta de línea de comandos *curl* para extraer el registro de Device Farm.

Durante o después de la sesión, puedes usar la `ListArtifacts`API de Device Farm para descargar el registro del servidor de Appium.

```
$ aws devicefarm list-artifacts \
```

```
  --type FILE \
  --arn arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678
```

Esto mostrará resultados como los siguientes durante la sesión:

```
{
    "artifacts": [
        {
            "arn": "arn:aws:devicefarm:us-
west-2:111122223333:artifact:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-4567
            "name": "AppiumServerLogOutput",
            "type": "APPIUM_SERVER_LOG_OUTPUT",
            "extension": "",
            "url": "https://prod-us-west-2-results.s3.dualstack.us-
west-2.amazonaws.com/111122223333/12345678..."
        }
    ]
}
```

Y lo siguiente una vez finalizada la sesión:

```
{
    "artifacts": [
        {
            "arn": "arn:aws:devicefarm:us-
west-2:111122223333:artifact:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-4567
            "name": "Appium Server Output",
            "type": "APPIUM_SERVER_OUTPUT",
            "extension": "log",
            "url": "https://prod-us-west-2-results.s3.dualstack.us-
west-2.amazonaws.com/111122223333/12345678..."
        }
    ]
}
```

```
$ curl "https://prod-us-west-2-results.s3.dualstack.us-
west-2.amazonaws.com/111122223333/12345678..."
```

Esto mostrará un resultado como el siguiente:

```
info Appium Welcome to Appium v2.5.4
```

```
info Appium Non-default server args:
info Appium { address: '127.0.0.1',
info Appium   allowInsecure:
info Appium    [ 'execute_driver_script',
info Appium      'session_discovery',
info Appium      'perf_record',
info Appium      'adb_shell',
info Appium      'chromedriver_autodownload',
info Appium      'get_server_logs' ],
info Appium   keepAliveTimeout: 0,
info Appium   logNoColors: true,
info Appium   logTimestamp: true,
info Appium   longStacktrace: true,
info Appium   sessionOverride: true,
info Appium   strictCaps: true,
info Appium   useDrivers: [ 'uiautomator' ] }
```

Python

Nota: en este ejemplo se utiliza el *`requests`* paquete de terceros para descargar el registro, así como el AWS SDK para Python*`boto3`*.

Durante o después de la sesión, puedes usar la `ListArtifacts`API de Device Farm para recuperar la URL del registro del servidor de Appium y, a continuación, descargarla.

```
import pathlib
import requests
import boto3

def download_appium_log():
    session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678
    client = boto3.client("devicefarm", region_name="us-west-2")

    # 1) List artifacts for the session (FILE artifacts), handling pagination
    artifacts = []
    token = None
    while True:
        kwargs = {"arn": session_arn, "type": "FILE"}
        if token:
            kwargs["nextToken"] = token
        resp = client.list_artifacts(**kwargs)
        artifacts.extend(resp.get("artifacts", []))
```

```
        token = resp.get("nextToken")
        if not token:
            break

    if not artifacts:
        raise RuntimeError("No artifacts found in this session")

    # Filter strictly to Appium server logs
    allowed = {"APPIUM_SERVER_OUTPUT", "APPIUM_SERVER_LOG_OUTPUT"}
    filtered = [a for a in artifacts if a.get("type") in allowed]
    if not filtered:
        raise RuntimeError("No Appium server log artifacts found (expected
 APPIUM_SERVER_OUTPUT or APPIUM_SERVER_LOG_OUTPUT)")

    # Prefer the final 'OUTPUT' log, else the live 'LOG_OUTPUT'
    chosen = (next((a for a in filtered if a.get("type") == "APPIUM_SERVER_OUTPUT"),
 None)
              or next((a for a in filtered if a.get("type") ==
 "APPIUM_SERVER_LOG_OUTPUT"), None))

    url = chosen["url"]
    ext = chosen.get("extension") or "log"
    out = pathlib.Path(f"./appium_server_log.{ext}")

    # 2) Download the artifact
    with requests.get(url, stream=True) as r:
        r.raise_for_status()
        with open(out, "wb") as fh:
            for chunk in r.iter_content(chunk_size=1024 * 1024):
                if chunk:
                    fh.write(chunk)

    print(f"Saved Appium server log to: {out.resolve()}")

download_appium_log()
```

Esto mostrará un resultado como el siguiente:

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', allowInsecure: [ 'execute_driver_script', ... ],
 useDrivers: [ 'uiautomator' ] }
```

Java

Nota: en este ejemplo se utiliza el AWS SDK para Java v2 y *`HttpClient`* para descargar el registro, y es compatible con las versiones 11 y superiores de JDK.

Durante la sesión o después de ella, puedes usar la `ListArtifacts`API de Device Farm para recuperar la URL del registro del servidor Appium y, a continuación, descargarla.

```
import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.nio.file.Path;
import java.time.Duration;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import software.amazon.awssdk.services.devicefarm.model.Artifact;
import software.amazon.awssdk.services.devicefarm.model.ArtifactCategory;
import software.amazon.awssdk.services.devicefarm.model.ListArtifactsRequest;
import software.amazon.awssdk.services.devicefarm.model.ListArtifactsResponse;

public class AppiumLogDownloader {

    public static void main(String[] args) throws Exception {
        String sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678

        try (DeviceFarmClient client = DeviceFarmClient.builder()
                .region(Region.US_WEST_2)
                .build()) {

            // 1) List artifacts for the session (FILE artifacts) with pagination
            List<Artifact> all = new ArrayList<>();
            String token = null;
            do {
                ListArtifactsRequest.Builder b = ListArtifactsRequest.builder()
                        .arn(sessionArn)
                        .type(ArtifactCategory.FILE);
```

```
                if (token != null) b.nextToken(token);
                ListArtifactsResponse page = client.listArtifacts(b.build());
                all.addAll(page.artifacts());
                token = page.nextToken();
            } while (token != null && !token.isBlank());

            // Filter strictly to Appium logs
            List<Artifact> filtered = all.stream()
                    .filter(a -> {
                        String t = a.typeAsString();
                        return "APPIUM_SERVER_OUTPUT".equals(t) ||
"APPIUM_SERVER_LOG_OUTPUT".equals(t);
                    })
                    .toList();

            if (filtered.isEmpty()) {
                throw new RuntimeException("No Appium server log artifacts found
(expected APPIUM_SERVER_OUTPUT or APPIUM_SERVER_LOG_OUTPUT).");
            }

            // Prefer OUTPUT; else LOG_OUTPUT
            Artifact chosen = filtered.stream()
                    .filter(a -> "APPIUM_SERVER_OUTPUT".equals(a.typeAsString()))
                    .findFirst()
                    .orElseGet(() -> filtered.stream()
                            .filter(a ->
"APPIUM_SERVER_LOG_OUTPUT".equals(a.typeAsString()))
                            .findFirst()
                            .get());

            String url = chosen.url();
            String ext = (chosen.extension() == null ||
chosen.extension().isBlank()) ? "log" : chosen.extension();
            Path out = Path.of("appium_server_log." + ext);

            // 2) Download the artifact with HttpClient
            HttpClient http = HttpClient.newBuilder()
                    .connectTimeout(Duration.ofSeconds(10))
                    .build();

            HttpRequest get = HttpRequest.newBuilder(URI.create(url))
                    .timeout(Duration.ofMinutes(5))
                    .GET()
                    .build();
```

```
            HttpResponse<Path> resp = http.send(get,
 HttpResponse.BodyHandlers.ofFile(out));
            if (resp.statusCode() / 100 != 2) {
                throw new IOException("Failed to download log, HTTP " +
 resp.statusCode());
            }
            System.out.println("Saved Appium server log to: " +
 out.toAbsolutePath());
        }
    }
}
```

Esto mostrará un resultado como el siguiente:

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', ..., useDrivers: [ 'uiautomator' ] }
```

JavaScript

Nota: en este ejemplo se utiliza el AWS SDK para JavaScript la versión 3 y el nodo 18+ *`fetch`* para descargar el registro.

Durante o después de la sesión, puedes usar la `ListArtifacts`API de Device Farm para recuperar la URL del registro del servidor de Appium y, a continuación, descargarla.

```
import { DeviceFarmClient, ListArtifactsCommand } from "@aws-sdk/client-device-
farm";
import { createWriteStream } from "fs";
import { pipeline } from "stream";
import { promisify } from "util";

const pipe = promisify(pipeline);
const client = new DeviceFarmClient({ region: "us-west-2" });

const sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678

// 1) List artifacts for the session (FILE artifacts), handling pagination
const artifacts = [];
let nextToken;
```

```
do {
  const page = await client.send(new ListArtifactsCommand({
    arn: sessionArn,
    type: "FILE",
    nextToken
  }));
  artifacts.push(...(page.artifacts ?? []));
  nextToken = page.nextToken;
} while (nextToken);

if (!artifacts.length) throw new Error("No artifacts found");

// Strict filter to Appium logs
const filtered = (artifacts ?? []).filter(a =>
  a.type === "APPIUM_SERVER_OUTPUT" || a.type === "APPIUM_SERVER_LOG_OUTPUT"
);
if (!filtered.length) {
  throw new Error("No Appium server log artifacts found (expected
 APPIUM_SERVER_OUTPUT or APPIUM_SERVER_LOG_OUTPUT).");
}

// Prefer OUTPUT; else LOG_OUTPUT
const chosen =
  filtered.find(a => a.type === "APPIUM_SERVER_OUTPUT") ??
  filtered.find(a => a.type === "APPIUM_SERVER_LOG_OUTPUT");

const url = chosen.url;
const ext = chosen.extension || "log";
const outPath = `./appium_server_log.${ext}`;

// 2) Download the artifact
const resp = await fetch(url);
if (!resp.ok) {
  throw new Error(`Failed to download log: ${resp.status} ${await
 resp.text().catch(()=>"")}`);
}
await pipe(resp.body, createWriteStream(outPath));
console.log("Saved Appium server log to:", outPath);
```

Esto mostrará un resultado como el siguiente:

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
```

```
info Appium { address: '127.0.0.1', allowInsecure: [ 'execute_driver_script', ... ],
 useDrivers: [ 'uiautomator' ] }
```

C#

Nota: en este ejemplo se utiliza el AWS SDK para .NET y *`HttpClient`* para descargar el registro.

Durante o después de la sesión, puedes usar la `ListArtifacts`API de Device Farm para recuperar la URL del registro del servidor de Appium y, a continuación, descargarla.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Net.Http;
using System.Threading.Tasks;
using System.Linq;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

class AppiumLogDownloader
{
    static async Task Main()
    {
        var sessionArn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678

        using var client = new AmazonDeviceFarmClient(RegionEndpoint.USWest2);

        // 1) List artifacts for the session (FILE artifacts), handling pagination
        var all = new List<Artifact>();
        string? token = null;
        do
        {
            var page = await client.ListArtifactsAsync(new ListArtifactsRequest
            {
                Arn = sessionArn,
                Type = ArtifactCategory.FILE,
                NextToken = token
            });
            if (page.Artifacts != null) all.AddRange(page.Artifacts);
            token = page.NextToken;
```

```
        } while (!string.IsNullOrEmpty(token));

        if (all.Count == 0)
            throw new Exception("No artifacts found");

        // Strict filter to Appium logs
        var filtered = all.Where(a =>
            a.Type == "APPIUM_SERVER_OUTPUT" || a.Type ==
 "APPIUM_SERVER_LOG_OUTPUT").ToList();

        if (filtered.Count == 0)
            throw new Exception("No Appium server log artifacts found (expected
 APPIUM_SERVER_OUTPUT or APPIUM_SERVER_LOG_OUTPUT).");

        // Prefer OUTPUT; else LOG_OUTPUT
        var chosen = filtered.FirstOrDefault(a => a.Type == "APPIUM_SERVER_OUTPUT")
                    ?? filtered.First(a => a.Type == "APPIUM_SERVER_LOG_OUTPUT");

        var url = chosen.Url;
        var ext = string.IsNullOrWhiteSpace(chosen.Extension) ? "log" :
 chosen.Extension;
        var outPath = $"./appium_server_log.{ext}";

        // 2) Download the artifact
        using var http = new HttpClient();
        using var resp = await http.GetAsync(url,
 HttpCompletionOption.ResponseHeadersRead);
        resp.EnsureSuccessStatusCode();
        await using (var fs = File.Create(outPath))
        {
            await resp.Content.CopyToAsync(fs);
        }
        Console.WriteLine($"Saved Appium server log to:
 {Path.GetFullPath(outPath)}");
    }
}
```

Esto mostrará un resultado como el siguiente:

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', ..., useDrivers: [ 'uiautomator' ] }
```

Ruby

Nota: en este ejemplo se usa el AWS SDK for Ruby y *`Net::HTTP`* se descarga el registro.

Durante o después de la sesión, puedes usar la `ListArtifacts`API de Device Farm para recuperar la URL del registro del servidor de Appium y, a continuación, descargarla.

```
require "aws-sdk-devicefarm"
require "net/http"
require "uri"

client = Aws::DeviceFarm::Client.new(region: "us-west-2")
session_arn = "arn:aws:devicefarm:us-
west-2:111122223333:session:12345678-1111-2222-333-456789abcdef/12345678-1111-2222-333-45678

# 1) List artifacts for the session (FILE artifacts), handling pagination
artifacts = []
token = nil
loop do
  page = client.list_artifacts(arn: session_arn, type: "FILE", next_token: token)
  artifacts.concat(page.artifacts || [])
  token = page.next_token
  break if token.nil? || token.empty?
end

raise "No artifacts found" if artifacts.empty?

# Strict filter to Appium logs
filtered = (artifacts || []).select { |a| ["APPIUM_SERVER_OUTPUT",
 "APPIUM_SERVER_LOG_OUTPUT"].include?(a.type) }
raise "No Appium server log artifacts found (expected APPIUM_SERVER_OUTPUT or
 APPIUM_SERVER_LOG_OUTPUT)." if filtered.empty?

# Prefer OUTPUT; else LOG_OUTPUT
chosen = filtered.find { |a| a.type == "APPIUM_SERVER_OUTPUT" } ||
         filtered.find { |a| a.type == "APPIUM_SERVER_LOG_OUTPUT" }

url = chosen.url
ext = (chosen.extension && !chosen.extension.empty?) ? chosen.extension : "log"
out_path = "./appium_server_log.#{ext}"

# 2) Download the artifact
uri = URI.parse(url)
Net::HTTP.start(uri.host, uri.port, use_ssl: (uri.scheme == "https")) do |http|
```

```
  req = Net::HTTP::Get.new(uri)
  http.request(req) do |resp|
    raise "Failed GET: #{resp.code} #{resp.body}" unless resp.code.to_i / 100 == 2
    File.open(out_path, "wb") { |f| resp.read_body { |chunk| f.write(chunk) } }
  end
end
puts "Saved Appium server log to: #{File.expand_path(out_path)}"
```

Esto mostrará un resultado como el siguiente:

```
info Appium Welcome to Appium v2.5.4
info Appium Non-default server args:
info Appium { address: '127.0.0.1', allowInsecure: [ 'execute_driver_script', ... ],
 useDrivers: [ 'uiautomator' ] }
```

Capacidades y comandos de Appium compatibles

El punto de conexión Appium de Device Farm admite la mayoría de los mismos comandos y capacidades deseadas que se utilizan en los dispositivos locales, con algunas excepciones. Las siguientes listas muestran qué funciones y comandos no se admiten actualmente. Si las pruebas no se pueden ejecutar según lo esperado debido a una capacidad restringida, abra un caso de soporte para obtener más información.

Capacidades compatibles

Al crear una sesión de Appium en Device Farm, recomendamos tener un conjunto distinto de capacidades que excluya las capacidades específicas de su dispositivo local. En Device Farm, la creación de sesiones puede fallar si se configuran determinadas capacidades no compatibles. Esto incluye capacidades específicas del dispositivo, como y. `udid` `platformVersion` Además, no se admiten determinadas funciones relacionadas ChromeDriver con WebDriverAgent Android y iOS, así como funciones que solo se admiten en emuladores y simuladores.

Comandos admitidos

La mayoría de los comandos de Appium que se ejecutan correctamente en dispositivos Android e iOS reales se ejecutarán según lo esperado en Device Farm, con las siguientes exclusiones:

Comandos de dispositivo Appium () **/appium/device**

- `install_app`
- `finger_print`
- `send_sms`
- `gsm_call`
- `gsm_signal`
- `gsm_voice`
- `power_ac`
- `power_capacity`
- `network_speed`
- `shake`

Appium ejecuta métodos y scripts () **/execute**

- `installApp`
- `execEmuConsoleCommand`
- `fingerprint`
- `gsmCall`
- `gsmSignal`
- `sendSms`
- `gsmVoice`
- `powerAC`
- `powerCapacity`
- `networkSpeed`
- `sensorSet`
- `injectEmulatorCameraImage`
- `isGpsEnabled`
- `shake`
- `clearApp`
- `clearKeychains`

- configureLocalization
- enrollBiometric
- getPasteboard
- installXCTestBundle
- listXCTestBundles
- listXCTestsInTestBundle
- runXCTest
- sendBiometricMatch
- setPasteboard
- setPermission
- startAudioRecording
- startLogsBroadcast
- startRecordingScreen
- startScreenStreaming
- startXCTestScreenRecording
- stopAudioRecording
- stopLogsBroadcast
- stopRecordingScreen
- stopScreenStreaming
- stopXCTestScreenRecording
- updateSafariPreferences

Dispositivos privados 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. Para obtener más información, consulte Creación de una ejecución de prueba o inicio de una sesión de acceso remoto en AWS Device Farm. 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 Creación de un perfil de instancia 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 conexión 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 - Heredados (no recomendado).

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

- Creación de un perfil de instancia en AWS Device Farm
- Solicitud de dispositivos privados adicionales en AWS Device Farm
- Creación de una ejecución de prueba o inicio de una sesión de acceso remoto en AWS Device Farm
- Selección de dispositivos privados en un grupo de dispositivos en AWS Device Farm

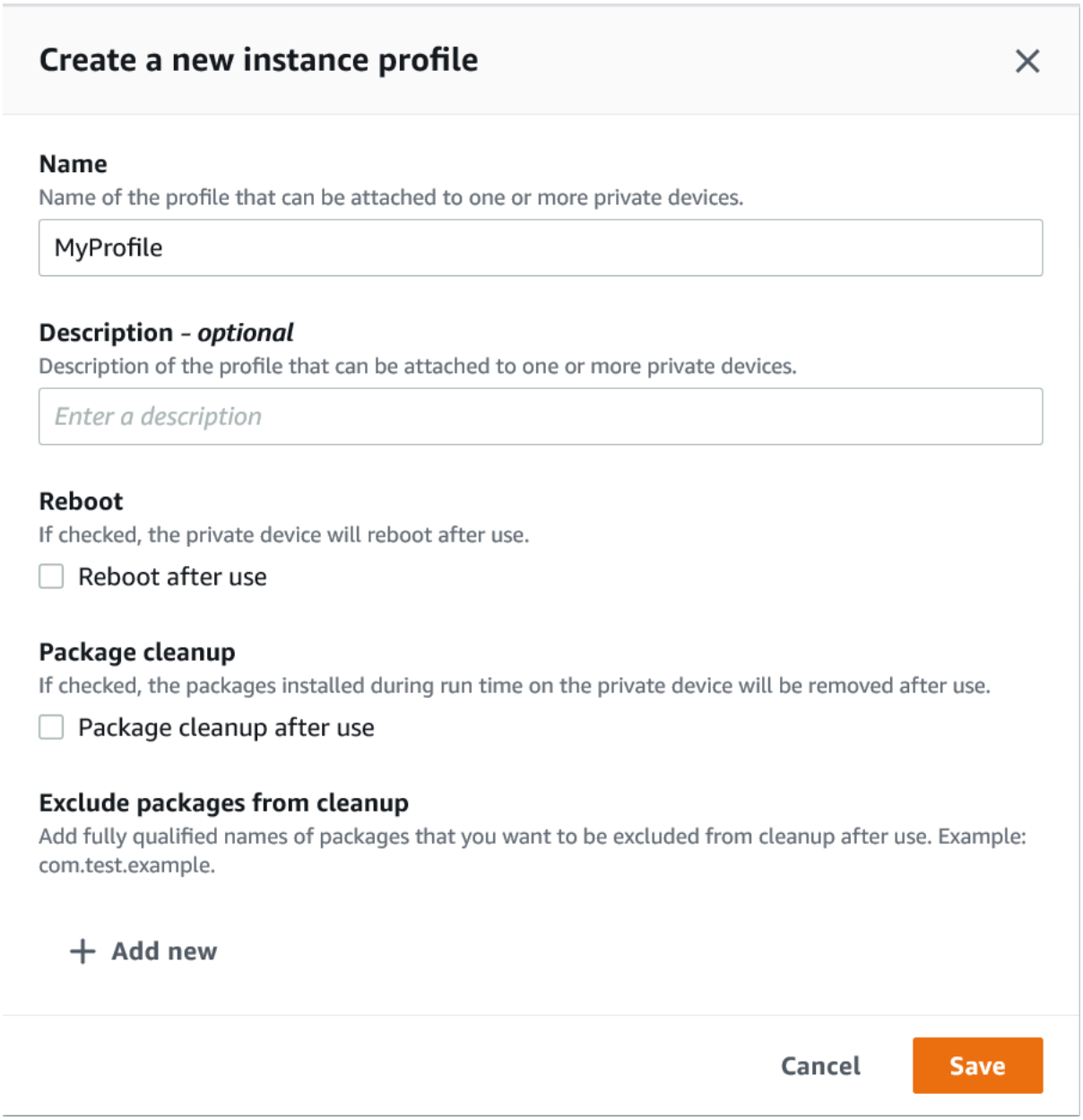
- Omitir la nueva firma de aplicación en dispositivos privados en AWS Device Farm
- Amazon VPC en todas AWS las regiones de AWS Device Farm
- Finalización de dispositivos privados en Device Farm

Creación de un perfil de instancia en AWS Device Farm

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. Los perfiles de instancia también pueden ayudarle a controlar el comportamiento de la sesión de acceso remoto. Para obtener más información acerca del uso de dispositivos privados en Device Farm, consulte Dispositivos privados en AWS Device Farm.

Para crear una instancia

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.

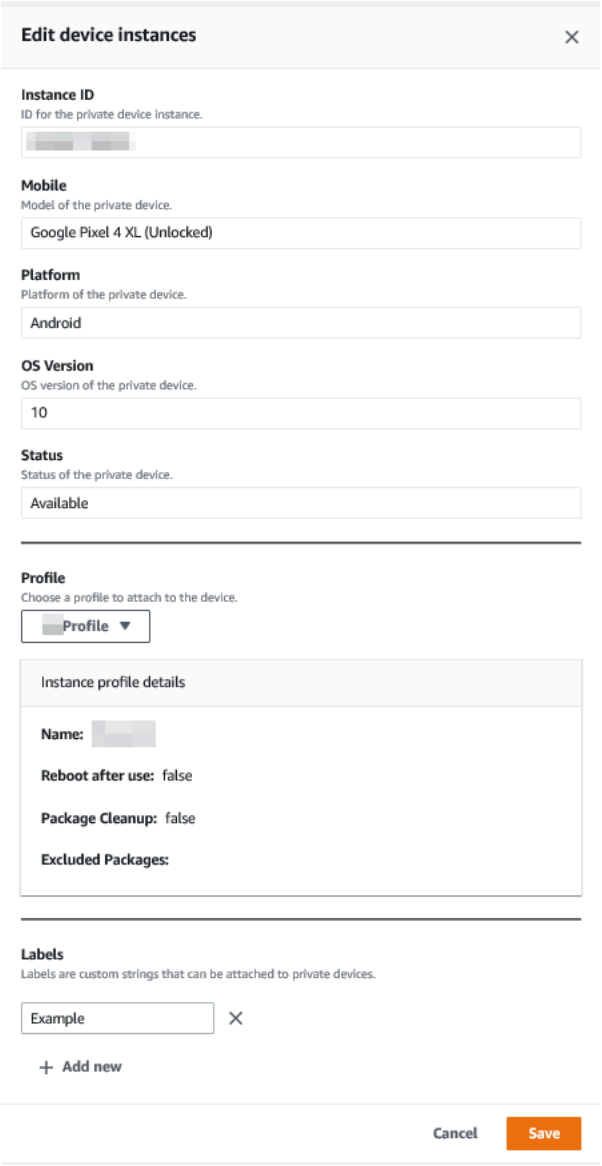
8. Seleccione Save.

Solicitud de dispositivos privados adicionales en AWS Device Farm

En AWS Device Farm, puede solicitar que se agreguen instancias de dispositivos privados adicionales a su flota. También puede ver y cambiar la configuración de las instancias de dispositivos privados existentes en su flota. Para obtener más información acerca de los dispositivos privados, consulte Dispositivos privados en AWS Device Farm.

Cómo solicitar dispositivos privados adicionales o cambiar su configuración

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. Para solicitar una nueva instancia de dispositivo privado, seleccione Solicitar instancia de dispositivo o Contáctenos. 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.



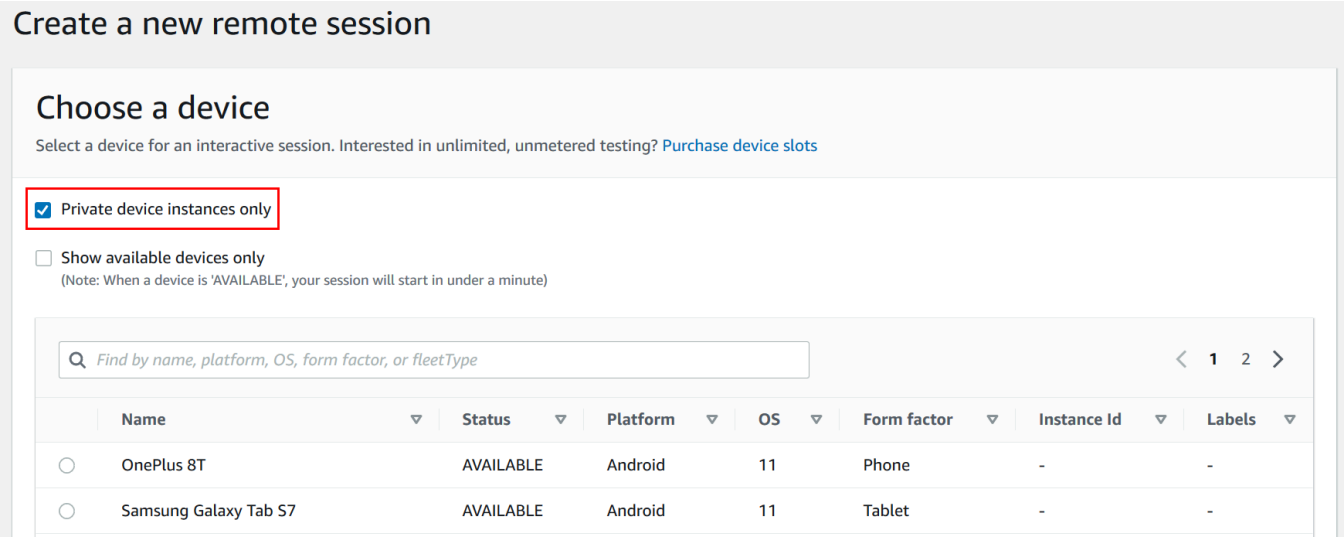
6. Para asociar un perfil de instancia a la instancia del dispositivo, elíjalo en el menú desplegable Perfil. Asociar un perfil de instancia puede resultar útil si, por ejemplo, desea excluir siempre un paquete de aplicaciones específico de las tareas de limpieza. Para obtener información sobre el uso de perfiles de instancia con dispositivos, consulte Creación de un perfil de instancia en AWS Device Farm.
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 en AWS Device Farm

En AWS Device Farm, 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. Para obtener más información acerca de los dispositivos privados, consulte Dispositivos privados en AWS Device Farm.

Cómo crear una ejecución de prueba o iniciar una sesión de acceso remoto

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. Realice 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, puedes editar un grupo de dispositivos existente o crear uno nuevo que incluya solo los dispositivos privados que el equipo de Device Farm configuró y asoció a tu AWS cuenta. 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 Elige un dispositivo, selecciona Instancias de dispositivos privados únicamente para limitar la lista a los dispositivos privados que el equipo de Device Farm configuró y asoció a tu AWS cuenta. 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.



Selección de dispositivos privados en un grupo de dispositivos en AWS Device Farm

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 interfaz de línea de AWS comandos (AWS CLI) o la API de Device Farm para crear o modificar un grupo de dispositivos con dispositivos privados mediante estas reglas.

Temas

- Crear un grupo de dispositivos privados con dispositivos privados (API)

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 tuvieras tres *`Apple iPhone 13`* dispositivos en la versión iOS *`16.1.0`* en tu 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 *iPhone 13* dispositivos *15.0.0* en el sistema operativo de su flota privada y cada dispositivo compartiera el mismo ARN de dispositivo, cada dispositivo también tendría su propio ARN de instancia que representaría únicamente 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 las etiquetas del dispositivo ARNs y de la instancia de dispositivo no se ajustan a su caso de uso. ARNs Las instancias de dispositivo suelen usarse como reglas para los 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 la instancia de dispositivo ARNs para su grupo de dispositivos, tendrá que cambiar manualmente la definición de la regla del grupo de dispositivos, pasando de usar el ARN anterior a usar el ARN nuevo. Si necesitas preconfigurar manualmente el dispositivo para su prueba, este puede ser un flujo de trabajo eficaz (en comparación con el dispositivo). ARNs 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.

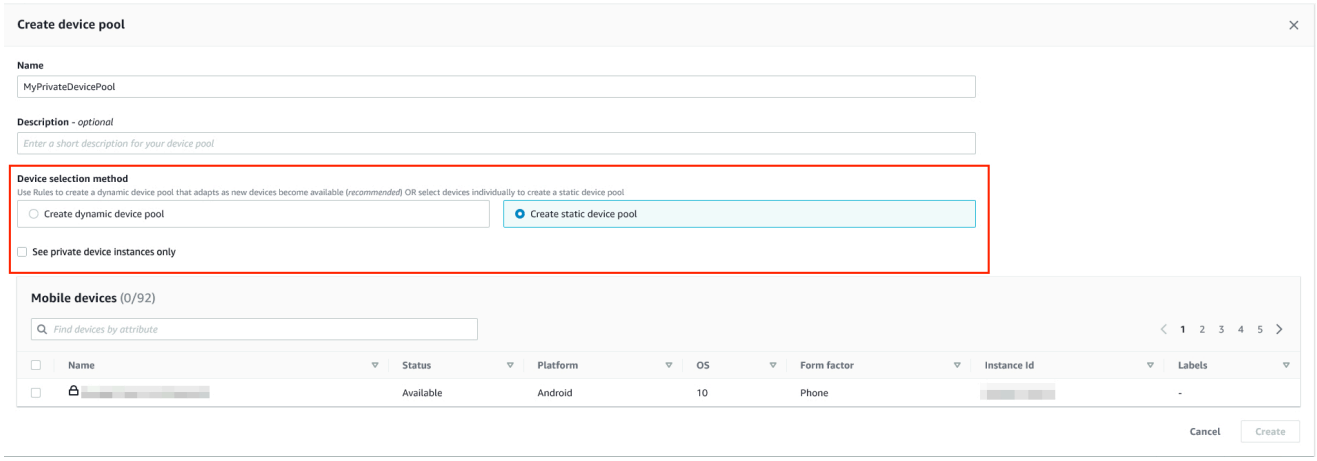
Crear un grupo de dispositivos privados con dispositivos privados (consola)

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.

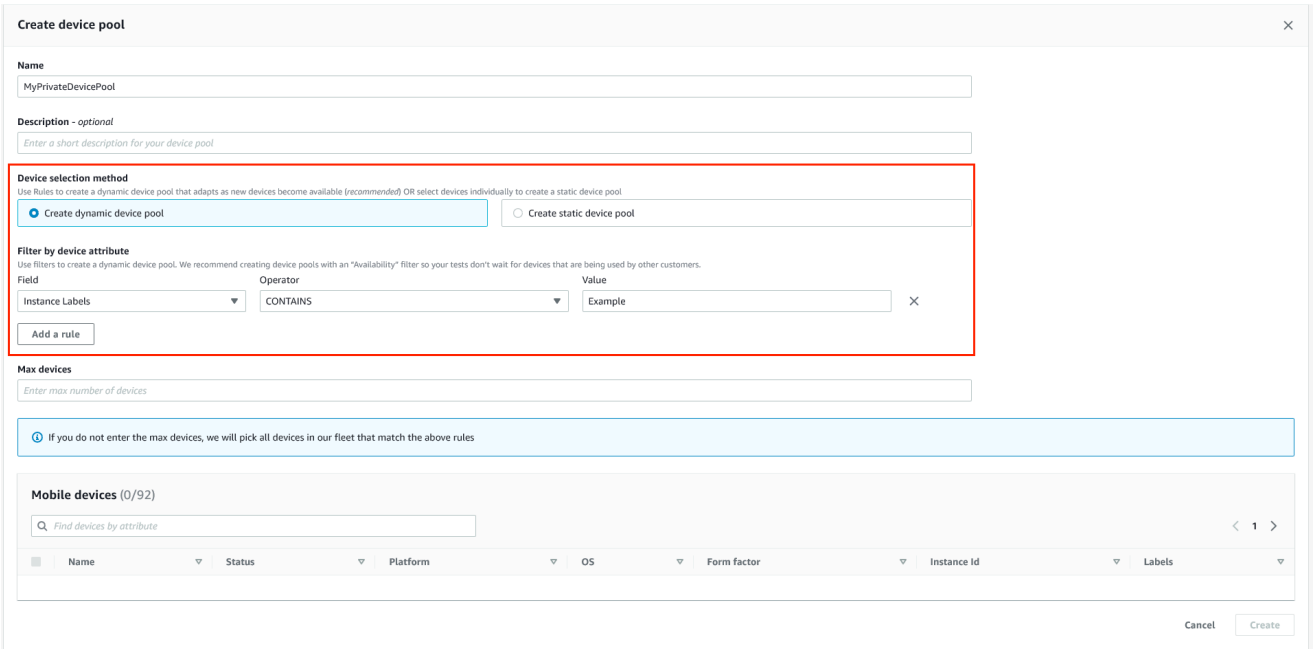
> **Note**
>
> Al crear un grupo de dispositivos con dispositivos privados en la consola, solo puede usar una de las tres reglas disponibles para seleccionar dispositivos privados. Si desea crear un grupo de dispositivos que contenga varios tipos de reglas para dispositivos privados (por ejemplo, grupos de dispositivos que contienen reglas para el dispositivo ARNs y la instancia de dispositivo ARNs), 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. Elija Configuración del proyecto y, a continuación, vaya a la pestaña Grupos de dispositivos.
5. En el paso Crear grupo de dispositivos, escriba un nombre y una descripción opcional para el grupo de dispositivos.

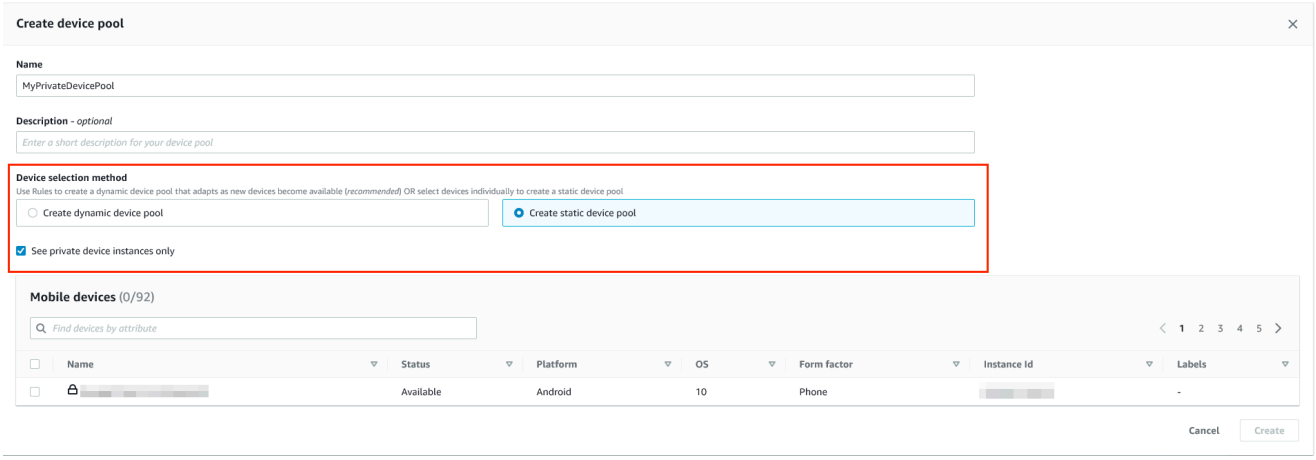
   a. Para usar las reglas de ARN de dispositivos para su grupo de dispositivos, seleccione Crear grupo de dispositivos estáticos y, a continuación, seleccione los tipos de dispositivos específicos de la lista que le gustaría usar en el grupo de dispositivos. No seleccione Solo instancias de dispositivos 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 su 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, elija Crear grupo de dispositivos estáticos y, a continuación, seleccione Instancias de dispositivos privados únicamente para limitar la lista de dispositivos a las instancias de dispositivos privados que Device Farm tenga asociadas a su AWS cuenta.



6. Seleccione Crear.

Crear un grupo de dispositivos privados con dispositivos privados (AWS CLI)

- Ejecute el comando create-device-pool.

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

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

La firma de aplicaciones es un proceso que implica la firma digital de un paquete de aplicaciones (por ejemplo, APK o IPA) con una clave privada antes de que pueda instalarse en un dispositivo o publicarse en una tienda de aplicaciones como Google Play Store o Apple App Store. Para agilizar las pruebas reduciendo el número de firmas y perfiles necesarios y aumentar la seguridad de los datos en los dispositivos remotos, AWS Device Farm volverá a firmar la aplicación después de que se cargue en el servicio.

Después de cargar la aplicación en AWS Device Farm, el servicio generará una nueva firma para la aplicación con sus propios certificados de firma y perfiles de aprovisionamiento. Este proceso reemplaza la firma original de la aplicación por la firma de AWS Device Farm. A continuación, la aplicación que se ha vuelto a firmar se instala en los dispositivos de prueba proporcionados por AWS Device Farm. La nueva firma permite instalar y ejecutar la aplicación en estos dispositivos sin los certificados originales del desarrollador.

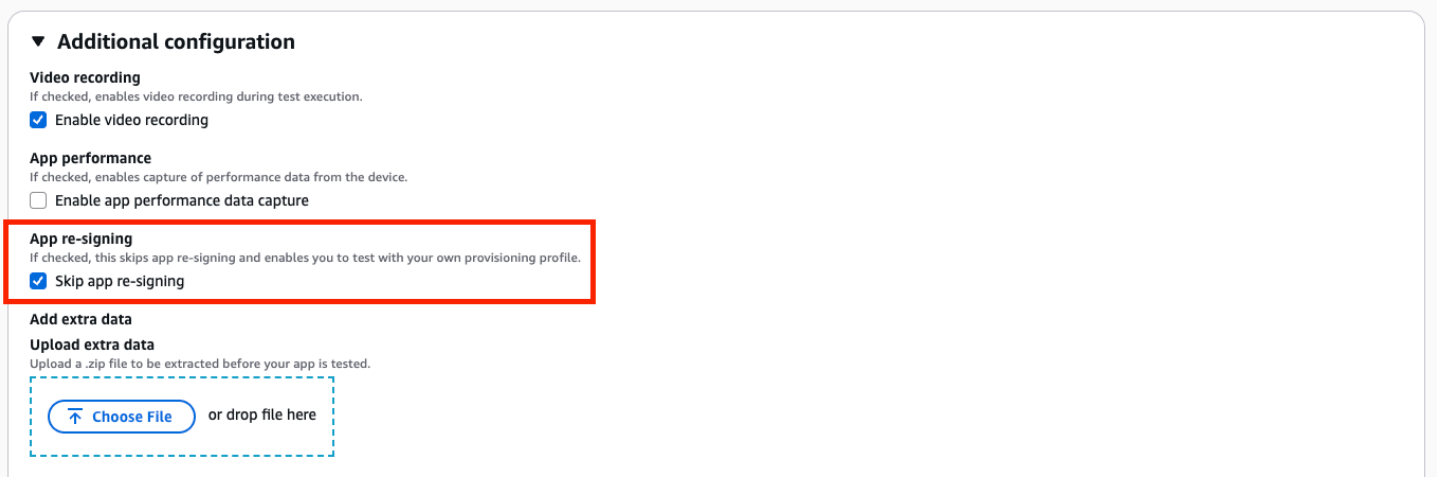
En iOS, sustituimos el perfil de aprovisionamiento integrado por un perfil comodín y volvemos a firmar la aplicación. Si proporciona datos auxiliares, los añadiremos al paquete de la aplicación antes de la instalación, de modo que esos datos estén presentes en el entorno de pruebas de la aplicación. Al volver a firmar la aplicación para iOS, se eliminarán todos los derechos.

En Android, volvemos a firmar la aplicación. Esto podría interrumpir las características que dependen de la firma de la aplicación, como la API de Google Maps para Android. También puede activar la detección antipiratería y antimanipulación, disponible en productos como. DexGuard Para las pruebas integradas, es posible que modifiquemos el manifiesto para incluir los permisos necesarios para capturar y guardar capturas de pantalla.

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 sobre los cambios que realiza Device Farm al probar la aplicación, consulte AWS Device Farm FAQs o la página de aplicaciones.

Para omitir que se vuelva a firmar las aplicaciones para una ejecución de prueba, seleccione Omitir volver a firmar aplicaciones en Configuración adicional. Esta opción solo está disponible para dispositivos privados.



> **Note**
>
> Si utiliza el XCTest marco, la opción Omitir la refirma de la aplicación no está disponible. Para obtener más información, consulte Integración de Device Farm con XCTest iOS.

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. Ve a la sección IDsCertificados 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 de 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.

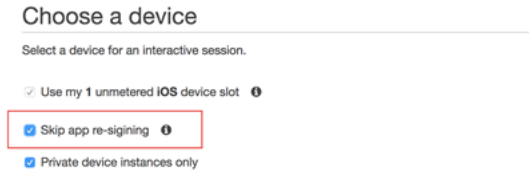
Para ello, debes instalar una aplicación de marcador de posición que esté firmada con el mismo certificado que la aplicación que quieres probar. Una vez que el dispositivo confíe en el perfil de configuración o en el desarrollador de la aplicación empresarial, todas las aplicaciones de ese desarrollador seguirán siendo de confianza en el dispositivo privado hasta que las elimines. Por lo tanto, cuando instales nuevas versiones de la aplicación que quieres probar, no tendrás que volver a confiar en el desarrollador de la aplicación cada vez. Esto resulta muy útil si ejecuta automatizaciones de prueba y no desea crear una sesión de acceso remoto cada vez que pruebe la aplicación.

Un procedimiento habitual que utilizan muchos clientes es volver a firmar la aplicación de ejemplo Device Farm para iOS y, a continuación, instalarla en su dispositivo como aplicación de marcador de posición.

Antes de iniciar la sesión de acceso remoto, siga los pasos en Creación de un perfil de instancia en AWS Device Farm para crear o modificar un perfil de instancia en Device Farm. En el perfil de la instancia, añade el ID de paquete de la aplicación de marcador de posición a la configuración 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.

Puedes cargar la aplicación de marcador de posición en el dispositivo mediante una sesión de acceso remoto, lo que te permite iniciar la aplicación y confiar en el desarrollador.

1. Siga las instrucciones de Creación de 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.

Asegúrate de añadir también la aplicación de marcador de posición o la aplicación que quieras probar a la configuración Excluir paquetes de la limpieza del perfil de instancia adjunto 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. Confirme que aparece un cuadro de diálogo de iOS que indica que el desarrollador de la aplicación empresarial no es de confianza.
5. Luego, si el dispositivo iOS tiene la versión 18 o superior de iOS, abre un ticket de soporte con el equipo de AWS Device Farm para que nuestro equipo confíe en la aplicación por ti, ya que estos dispositivos requieren que la aplicación sea confiable manualmente. De lo contrario, si la versión de iOS es 17 o inferior, puedes ir a la aplicación Configuración y, en Configuración general, confiar tú mismo en la aplicación desde el menú VPN y Perfiles.

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.

Amazon VPC en todas AWS las regiones de AWS Device Farm

Los servicios de Device Farm solo se encuentran en la región Oeste de EE. UU. (Oregón) (`us-west-2`). Puede utilizar Amazon Virtual Private Cloud (Amazon VPC) para acceder a un servicio de su Amazon Virtual Private Cloud en 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 - Heredados (no recomendado).

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.

Información general sobre la interconexión de VPC VPCs en diferentes regiones en AWS Device Farm

Puede comparar dos de ellas VPCs en regiones diferentes, siempre que tengan bloques CIDR distintos y que no se superpongan. Esto garantiza que todas las direcciones IP privadas sean únicas y permite que todos los recursos de la misma se direccionen entre sí sin necesidad de ningún tipo de traducción de direcciones de red (NAT). VPCs Para obtener más información acerca de la notación CIDR, consulte RFC 4632.

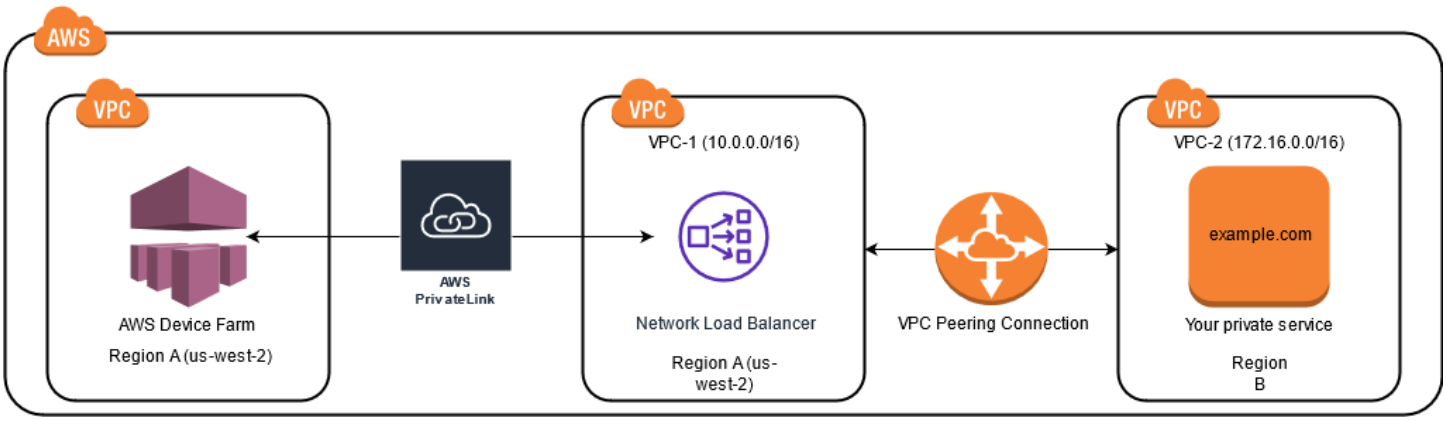
En este tema se incluye un escenario de ejemplo entre regiones en el que Device Farm (denominado VPC-1) se encuentra en la región de Oeste de EE. UU. (Oregón) (`us-west-2`). La segunda VPC del ejemplo se encuentra en otra región y se denomina VPC-2.

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 emparejamiento entre dos VPCs puede cambiar la postura de seguridad del VPCs. Además, agregar nuevas entradas a sus tablas de enrutamiento puede cambiar la postura de seguridad de los recursos incluidos en. VPCs 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 para usar Amazon VPC en AWS Device Farm
- Paso 1: configuración de una conexión de emparejamiento entre VPC-1 y VPC-2
- Paso 2: actualización de las tablas de enrutamiento en VPC-1 y VPC-2
- Paso 3: creación de un grupo de destino
- Paso 4: Crear un Network Load Balancer
- Paso 5: creación de un servicio de punto de conexión de VPC para conectar su VPC a Device Farm
- Paso 6: creación de una configuración de punto de conexión de VPC entre su VPC y Device Farm
- Paso 7: creación de una ejecución de prueba para usar la configuración de punto de conexión de VPC
- Creación de una red escalable con una puerta de enlace de tránsito

Requisitos previos para usar Amazon VPC en AWS Device Farm

Este ejemplo requiere lo siguiente:

- Dos VPCs que están 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 más información sobre la creación VPCs y configuración de subredes, consulte Cómo trabajar con subredes VPCs y subredes en la Guía de peering de Amazon VPC.

Paso 1: configuración de una conexión de emparejamiento entre VPC-1 y VPC-2

Establezca una conexión de emparejamiento entre los dos bloques CIDR VPCs 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
```

Region

```
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: actualización de las tablas de enrutamiento en VPC-1 y VPC-2

Después de configurar una conexión de interconexión, debe establecer una ruta de destino entre las dos VPCs 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: creación de 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 y, a continuación, seleccione Instances. https://console.aws.amazon.com/ec2/
     > - 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 y https://console.aws.amazon.com/ec2/, a continuación, elegir 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.

     Como IPv4 dirección, elija la dirección IP de la VPC-2.

     En el caso de Puertos, elija los suyos.
   - Elija 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
```

IPv4 address

```
172.16.100.60
```

Puertos

```
80
```

Paso 4: Crear un Network Load Balancer

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: creación de un servicio de punto de conexión de VPC para conectar su VPC a Device Farm

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: creación de una configuración de punto de conexión de VPC entre su VPC y 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: creación de una ejecución de prueba para usar la configuración de punto de conexión de VPC

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 Creación de una ejecución de prueba en Device Farm o Creación de una sesión.

Creación de una red escalable con una puerta de enlace de tránsito

Para crear una red escalable con más de dos VPCs, puede usar Transit Gateway como centro de tránsito de red para interconectar sus VPCs redes con las locales. 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.

Finalización de dispositivos privados en Device Farm

Para cancelar un dispositivo privado después del plazo inicialmente acordado, debes avisar con 30 días de antelación si no lo renueva a través de nuestro correo electrónico `<aws-devicefarm-support@amazon>` .com. Para obtener más información acerca de los dispositivos privados, consulte Dispositivos privados en AWS Device Farm.

> **Important**
>
> Estas instrucciones solo se aplican a la finalización 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 AWS el servicio de asistencia.

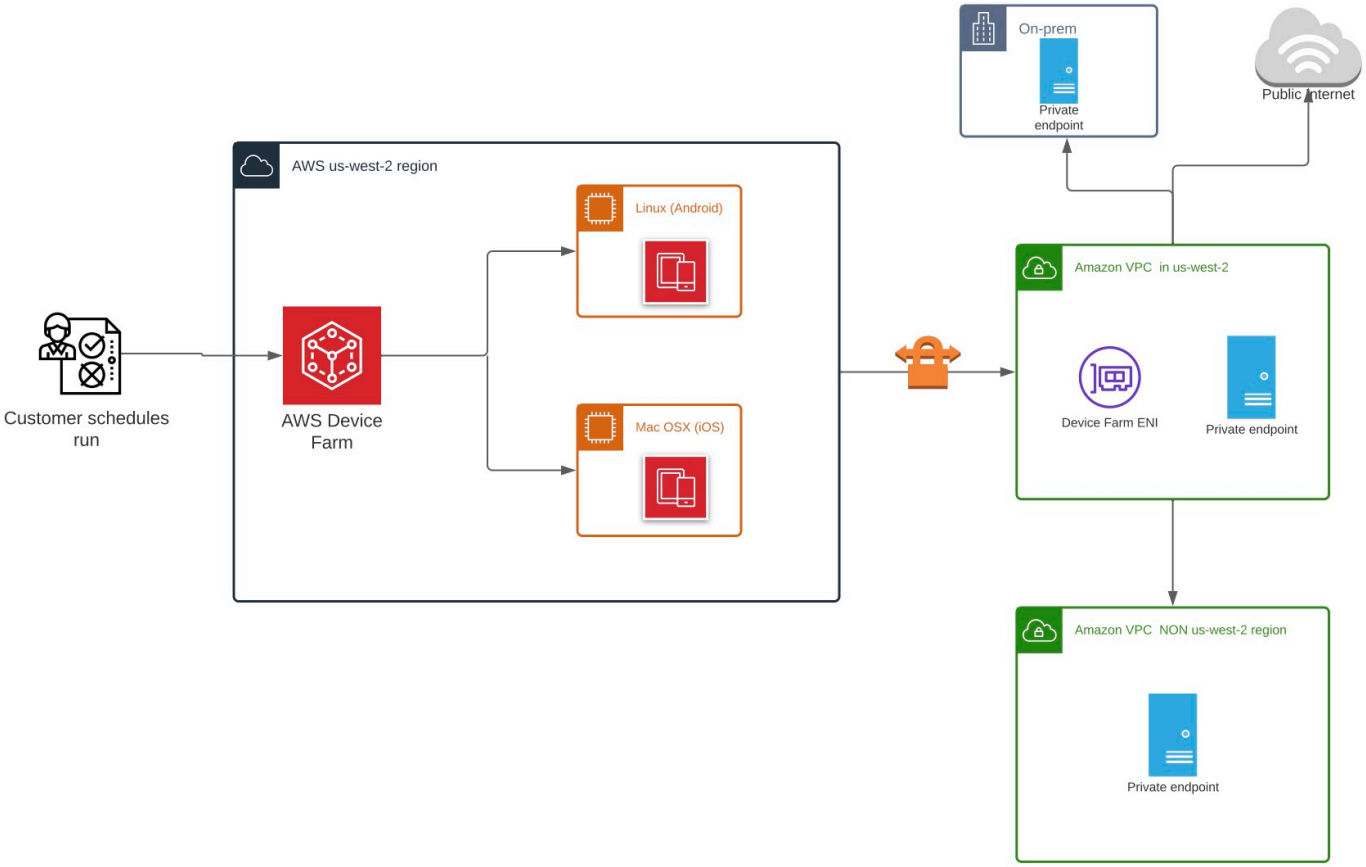
VPC-ENI en AWS Device Farm

> **Warning**
>
> Esta característica solo está disponible en dispositivos privados. Para solicitar el uso de un dispositivo privado en tu AWS cuenta, ponte en contacto con nosotros. Si ya tienes dispositivos privados agregados a tu AWS cuenta, te recomendamos encarecidamente que utilices este método de conectividad de VPC.

La función de conectividad VPC-ENI de AWS Device Farm ayuda a los clientes a conectarse de forma segura a sus puntos de conexión privados alojados en AWS un software local o a otro proveedor de nube.

Puede conectar los dispositivos móviles Device Farm y sus máquinas host a un entorno de Amazon Virtual Private Cloud (Amazon VPC) en la `us-west-2` región, lo que permite el acceso a non-internet-facing servicios y aplicaciones aislados a través de una interface de red elástica. Para obtener más información VPCs, 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 AWS CloudFormation cómo crear y comparar automáticamente VPCs, consulte las VPCPeering plantillas del repositorio de AWS CloudFormation plantillas en GitHub.

> **Note**
>
> Device Farm no cobra nada por crear ENIs en la VPC de un cliente en. `us-west-2` El costo 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, públicos CDNs) 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

AWS control de acceso e IAM

AWS Device Farm le permite usar AWS Identity and Access Management (IAM) para crear políticas que concedan o restrinjan el acceso a los roles 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:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [{
      "Effect": "Allow",
      "Action": [
        "devicefarm:*",
        "ec2:DescribeVpcs",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
        "ec2:CreateNetworkInterface"
      ],
      "Resource": [
        "*"
      ]
    },
    {
```

```
      "Effect": "Allow",
      "Action": "iam:CreateServiceLinkedRole",
      "Resource": "arn:aws:iam::*:role/aws-service-role/devicefarm.amazonaws.com/
AWSServiceRoleForDeviceFarm",
      "Condition": {
        "StringLike": {
          "iam:AWSServiceName": "devicefarm.amazonaws.com"
        }
      }
    }
  ]
}
```

Para 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 funciones AWS Identity and Access Management vinculadas a servicios (IAM). Un rol vinculado a un servicio es un tipo único de rol de IAM que está vinculado directamente a Device Farm. Device Farm predefine las funciones vinculadas al servicio e incluyen todos los permisos que el servicio requiere para llamar a otros AWS servicios 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 es posible eliminar un rol vinculado 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 utiliza el rol vinculado al servicio denominado AWSServiceRoleForDeviceFarm: Permite que Device Farm acceda a los recursos de AWS en su nombre.

El rol AWSService RoleForDeviceFarm vinculado al servicio confía en 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
  - Describa VPCs
  - 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:

JSON

```
{
"Version":"2012-10-17",
"Statement": [
 {
  "Effect": "Allow",
  "Action": [
   "ec2:DescribeNetworkInterfaces",
   "ec2:DescribeVpcs",
   "ec2:DescribeSubnets",
   "ec2:DescribeSecurityGroups"
  ],
  "Resource": "*"
 },
 {
  "Effect": "Allow",
  "Action": [
   "ec2:CreateNetworkInterface"
  ],
  "Resource": [
   "arn:aws:ec2:*:*:subnet/*",
   "arn:aws:ec2:*:*:security-group/*"
  ]
 },
 {
  "Effect": "Allow",
  "Action": [
   "ec2:CreateNetworkInterface"
  ],
  "Resource": [
   "arn:aws:ec2:*:*:network-interface/*"
  ],
  "Condition": {
   "StringEquals": {
    "aws:RequestTag/AWSDeviceFarmManaged": "true"
   }
```

```
    }
   },
   {
    "Effect": "Allow",
    "Action": [
     "ec2:CreateTags"
    ],
    "Resource": "arn:aws:ec2:*:*:network-interface/*",
    "Condition": {
     "StringEquals": {
      "ec2:CreateAction": "CreateNetworkInterface"
     }
    }
   },
   {
    "Effect": "Allow",
    "Action": [
     "ec2:CreateNetworkInterfacePermission",
     "ec2:DeleteNetworkInterface"
    ],
    "Resource": "arn:aws:ec2:*:*:network-interface/*",
    "Condition": {
     "StringEquals": {
      "aws:ResourceTag/AWSDeviceFarmManaged": "true"
     }
    }
   },
   {
    "Effect": "Allow",
    "Action": [
     "ec2:ModifyNetworkInterfaceAttribute"
    ],
    "Resource": [
     "arn:aws:ec2:*:*:security-group/*",
     "arn:aws:ec2:*:*:instance/*"
    ]
   },
   {
    "Effect": "Allow",
    "Action": [
     "ec2:ModifyNetworkInterfaceAttribute"
    ],
    "Resource": "arn:aws:ec2:*:*:network-interface/*",
    "Condition": {
```

```
    "StringEquals": {
     "aws:ResourceTag/AWSDeviceFarmManaged": "true"
    }
   }
  }
 ]
}
```

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 crea su primer recurso de Device Farm en la Consola de administración de AWS, la AWS CLI o la AWS API, Device Farm crea el rol vinculado al servicio por usted.

Si elimina este rol vinculado a servicios 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 en la AWS API, cree una función vinculada al servicio con el nombre del servicio. `devicefarm.amazonaws.com` Para obtener más información, consulte Creación de 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 permite editar el rol AWSService RoleForDeviceFarm vinculado al servicio. Después de crear un rol vinculado al servicio, no podrá cambiar el nombre del rol, ya que varias entidades podrían hacer referencia al rol. Sin embargo, sí puede editar la descripción del rol con IAM. Para obtener más información, consulte Edición de 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 usar una característica o servicio que requieran un rol vinculado a un servicio, le recomendamos que elimine dicho rol. Así no tendrá una entidad no utilizada que no se supervise ni mantenga de forma activa. Sin embargo, debe limpiar los recursos de su 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 AWS CLI, la o la AWS API para eliminar la función vinculada al AWSService RoleForDeviceFarm servicio. 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 conexión 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 usar el AWSService RoleForDeviceFarm rol en las siguientes regiones.

Nombre de la región	Identidad de la región	Compatibilidad en Device Farm
Este de EE. UU. (Norte de Virginia)	us-east-1	No
Este de EE. UU. (Ohio)	us-east-2	No
Oeste de EE. UU. (Norte de California)	us-west-1	No

Nombre de la región	Identidad de la región	Compatibilidad en Device Farm
Oeste de EE. UU. (Oregón)	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
Asia-Pacífico (Singapur)	ap-southeast-1	No
Asia-Pacífico (Sídney)	ap-southeast-2	No
Asia-Pacífico (Tokio)	ap-northeast-1	No
Canadá (centro)	ca-central-1	No
Europa (Fráncfort)	eu-central-1	No
Europa (Irlanda)	eu-west-1	No
Europa (Londres)	eu-west-2	No
Europa (París)	eu-west-3	No
América del Sur (São Paulo)	sa-east-1	No
AWS GovCloud (US)	us-gov-west-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 AWS cuenta.
- Debe tener un usuario o un rol de AWS cuenta con permisos para crear un rol vinculado al Servicio. Cuando se utilizan puntos de enlace de Amazon VPC con las funciones de pruebas

móviles de Device Farm, Device Farm crea un rol vinculado al servicio AWS Identity and Access Management (IAM).

- Device Farm VPCs solo se puede conectar en la `us-west-2` región. 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 la interconexión VPCs, 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`:
  - ID de VPC
  - Subred IDs (solo subredes privadas)
  - Grupo de seguridad IDs
- 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 los roles 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.
- Al utilizar VPC-ENI, el solucionador de DNS que utilizan los hosts y dispositivos de prueba de Device Farm será el servidor proporcionado por los servicios de DHCP en la subred del cliente. En una configuración predeterminada, será el solucionador predeterminado de la VPC. Los clientes que deseen especificar solucionadores de DNS personalizados pueden configurar un conjunto de opciones de DHCP en su 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. Seleccione Configuración del proyecto.
5. En la sección Configuración de nube privada virtual (VPC), puede cambiar la `VPC`, las `Subnets` (solo subredes privadas) y los `Security Groups`.
6. Seleccione Save.

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 disponibles. IPv4
- 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.
- La integración de VPC-ENI solo es compatible con las subredes privadas de la VPC.
- 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.

Uso de los servicios de puntos de conexión de VPC de Amazon con Device Farm - Heredados (no recomendado)

> **Warning**
>
> Recomendamos encarecidamente utilizar la conectividad VPC-ENI que se describe en esta página para la conectividad de puntos de conexión privados, ya que la VPCE ahora se considera una característica heredada. VPC-ENI proporciona más flexibilidad,

configuraciones más sencillas, es más rentable y supone un gasto de mantenimiento significativamente menor en comparación con el método de conectividad VPCE.

> **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 AWS servicio que puede utilizar para lanzar AWS recursos en una red virtual que usted 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 para alojar aplicaciones privadas en la AWS región EE.UU. Oeste (Oregón) (`us-west-2`), 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 permitir que su AWS cuenta utilice esta función con dispositivos privados, póngase en contacto 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 los servicios de punto final de VPC PrivateLink (AWS) en la AWS PrivateLink guía.

> **Important**
>
> La función de punto final de la 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. AWS PrivateLink Aunque la conexión es segura y privada, la seguridad depende de la protección de sus credenciales de AWS . Si sus AWS credenciales están

comprometidas, un atacante puede acceder a los datos de su servicio o exponerlos al mundo exterior.

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 VPC 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 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 Oyentes 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 equilibrio 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 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 equilibrador de carga, en Configuración básica, haga lo siguiente:
   a. Introduzca un Nombre para el equilibrador de carga.
   b. En Esquema, seleccione Interno.
6. En Oyentes, 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 equilibrio 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. Realice 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 conexión 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 conexión de la VPC en el siguiente paso.

Su servicio de punto de conexión de VPC ya está listo para utilizarlo con Device Farm.

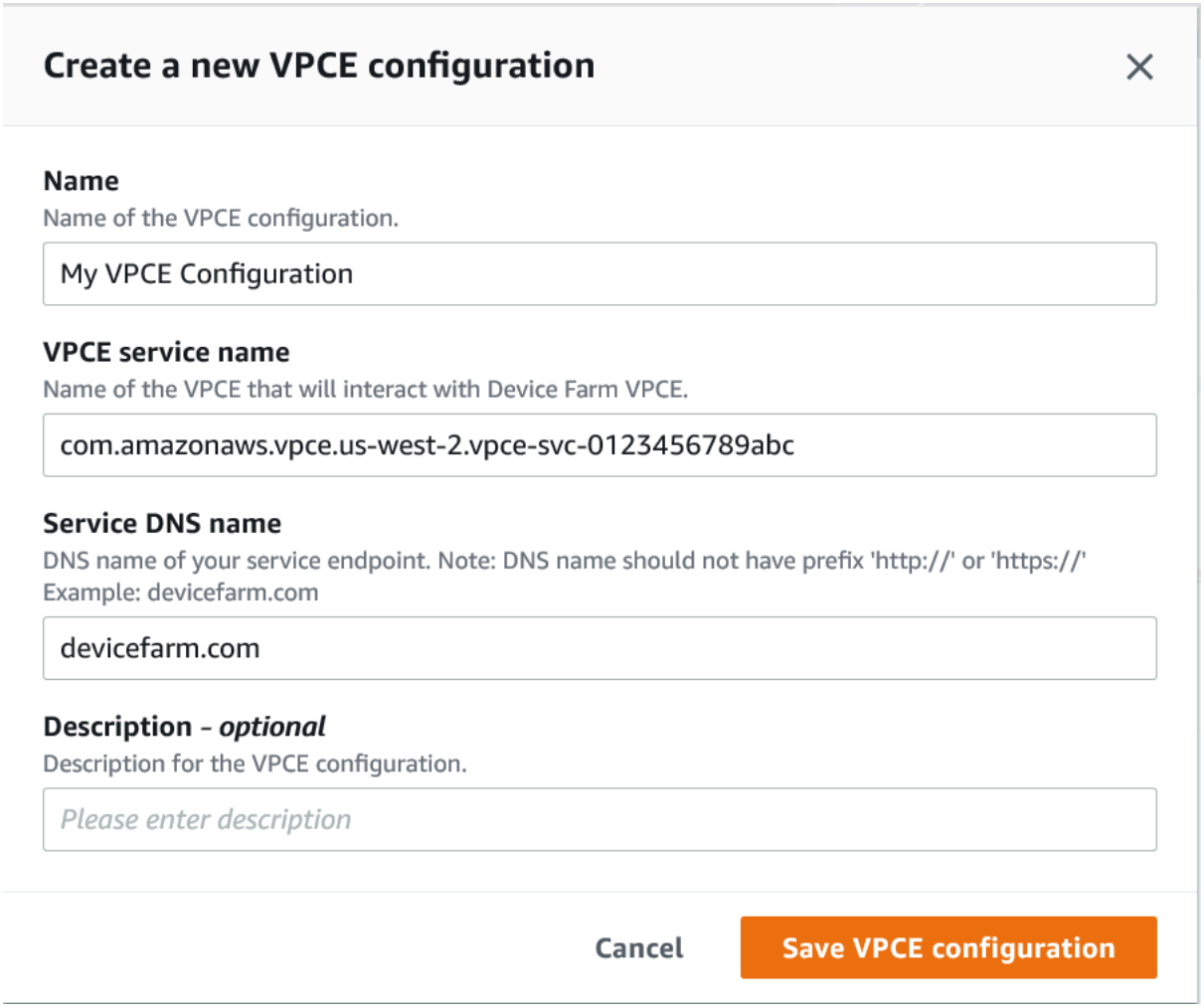
Paso 3: Crear una configuración de punto de conexión de VPC en Device Farm

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 Creación de una ejecución de prueba en Device Farm o Creación de una sesión.

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 sobre 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 Visualización de 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 más información, consulte los siguientes temas:

- Introducción a la creación de registros de seguimiento
- Consulte Servicios e integraciones compatibles con CloudTrail
- Configuración de notificaciones de Amazon SNS para CloudTrail

- Recepción de archivos de registro de CloudTrail de varias regiones y Recepción de 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 AWS CLIReferencia de 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 lo ayuda a determinar lo siguiente:

- Si la solicitud se realizó con credenciales de usuario de 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 de registro de CloudTrail pueden contener una o varias entradas de registro. Un evento representa una solicitud específica realizada desde un origen cualquiera 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:

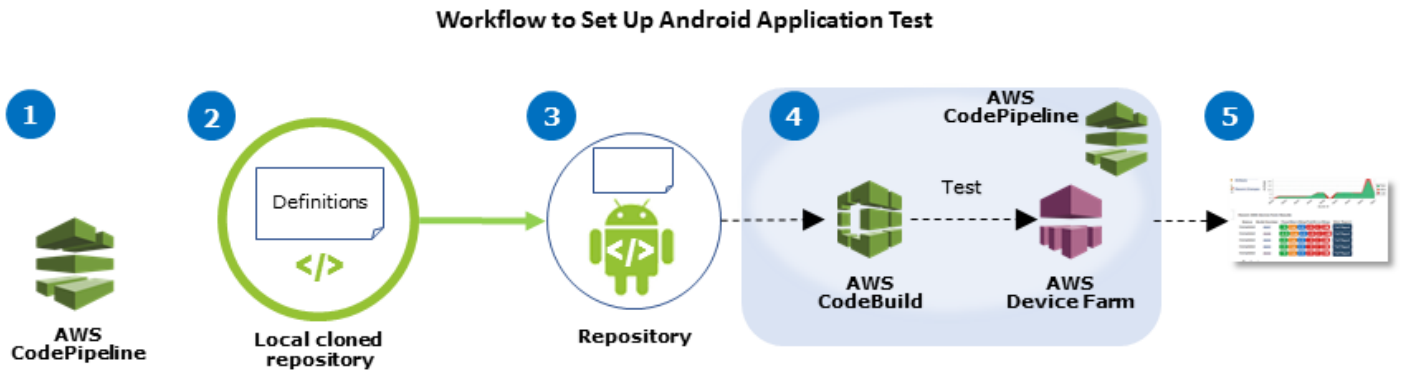
```
{
  "Records": [
    {
      "eventVersion": "1.03",
      "userIdentity": {
        "type": "Root",
        "principalId": "AKIAI44QH8DHBEXAMPLE",
        "arn": "arn:aws:iam::123456789012:root",
        "accountId": "123456789012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "sessionContext":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

Integración de AWS Device Farm en una fase CodePipeline de prueba

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 el tutorial: Compila y prueba una aplicación de Android cuando se envía a ella 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

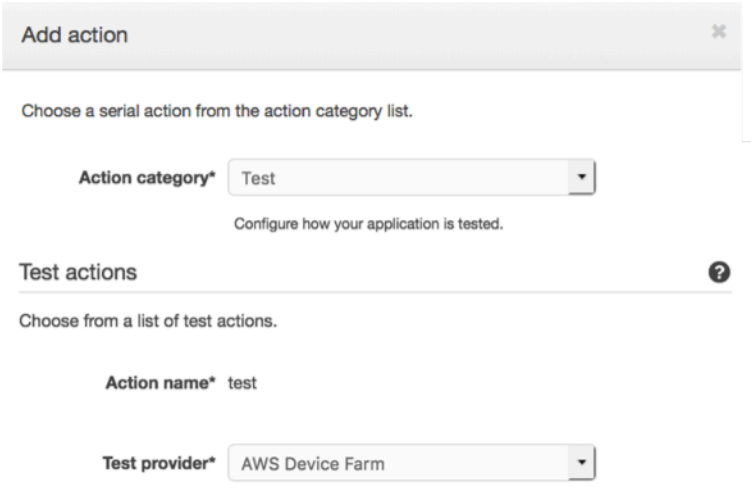
Para obtener información sobre cómo configurar una canalización que prueba continuamente una aplicación compilada (como un archivo `.ipa` de iOS o `.apk` de Android) como su origen, consulte Tutorial: Crear una canalización que compile y pruebe la aplicación iOS después de un cambio en el bucket de Amazon 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.

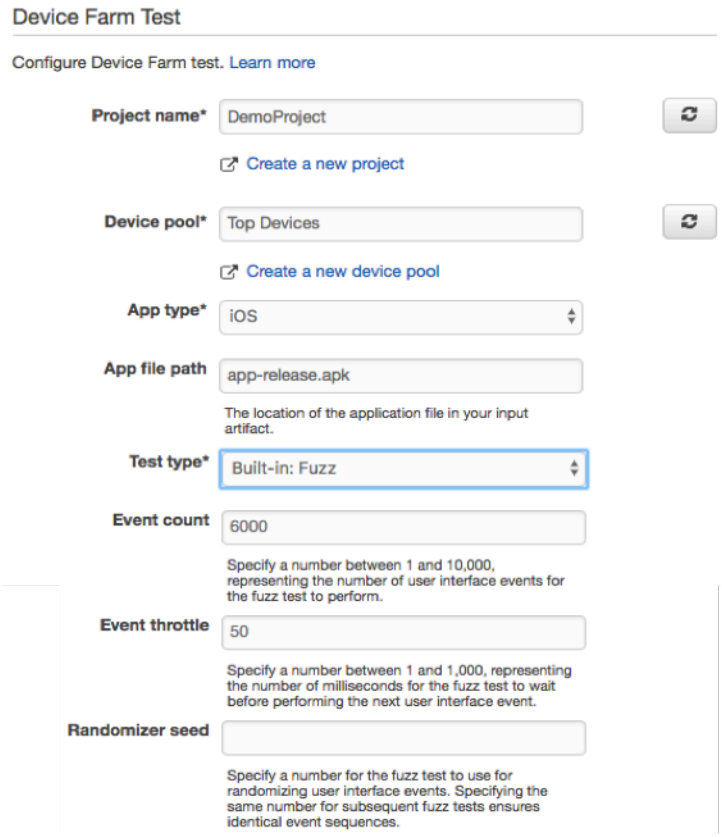
Para configurar una ejecución de prueba de Device Farm como acción CodePipeline de prueba

1. Inicie sesión en Consola de administración de AWS y abra la CodePipeline consola 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.



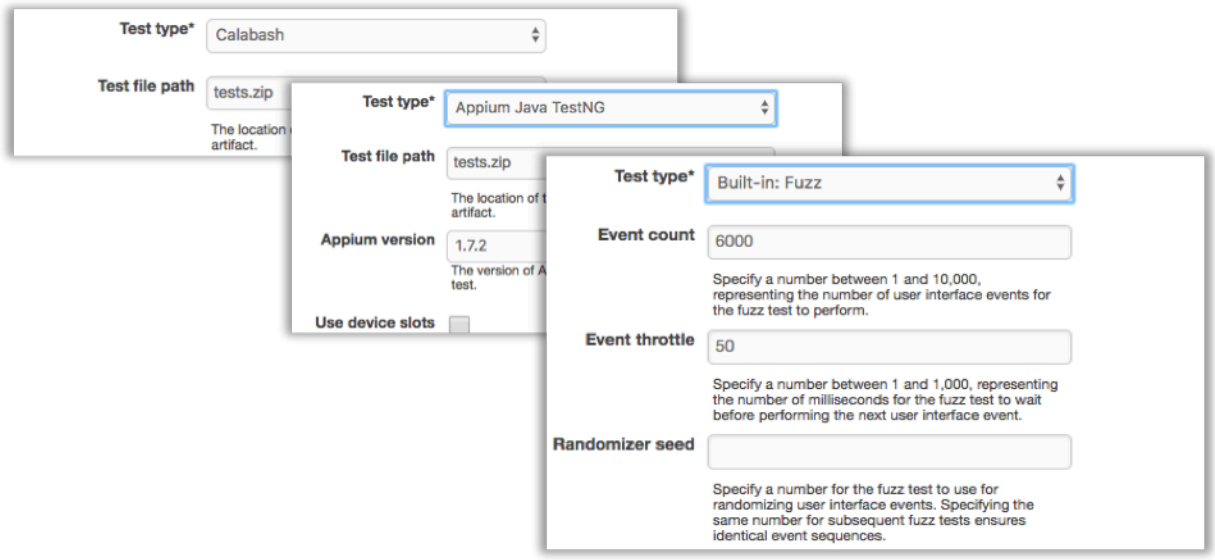
7. En Nombre del proyecto, seleccione su proyecto de Device Farm existente o seleccione Crear un nuevo proyecto.

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.

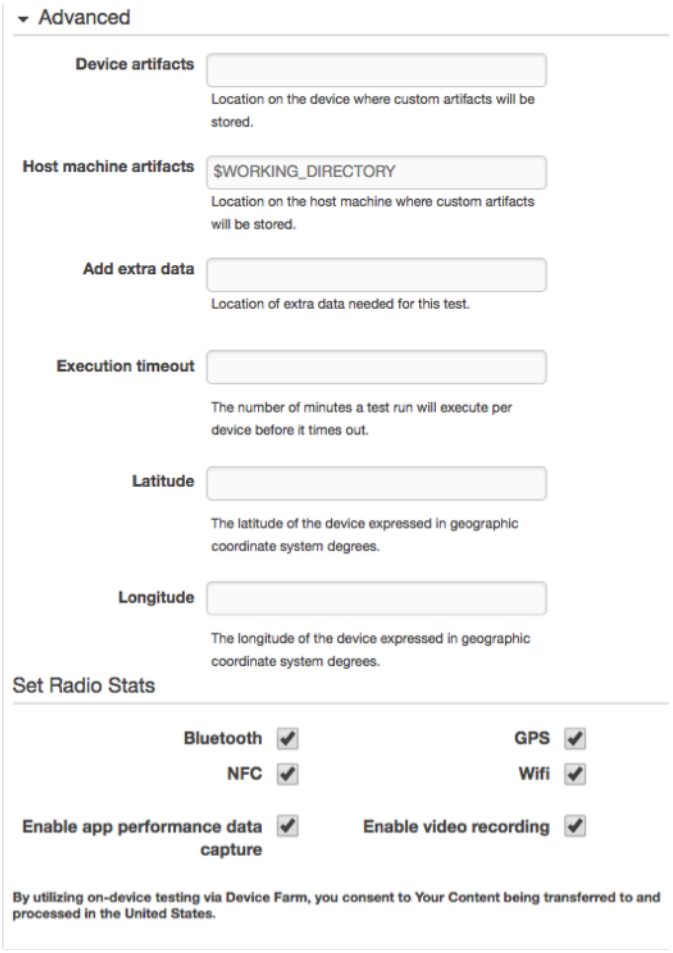


10. En Ruta de archivo de aplicación, escriba la ruta del paquete de aplicación compilado. La ruta es relativa a la raíz del artefacto de entrada de la prueba.
11. En Tipo de prueba, realice alguna de las siguientes operaciones:

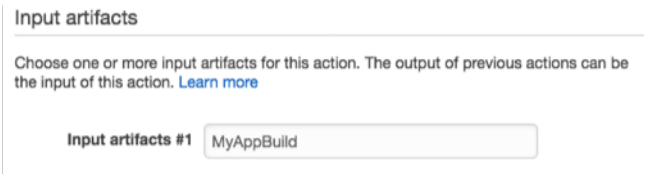
    - Si utiliza una de las pruebas de Device Farm integradas, elija el tipo de la prueba configurada en su proyecto de Device Farm.
    - Si no utiliza una de las pruebas integradas de Device Farm, en Ruta de archivo de prueba escriba la ruta del archivo de definición de prueba. La ruta es relativa a la raíz del artefacto de entrada de la prueba.



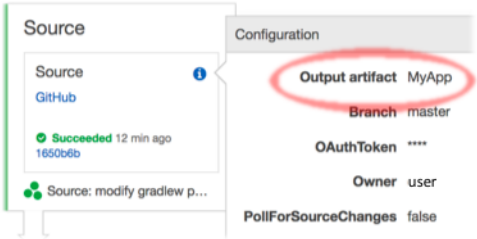
12. En los campos restantes, proporcione la configuración que sea adecuada para su prueba y tipo de aplicación.
13. (Opcional) En Avanzado, proporcione una configuración detallada de la ejecución de prueba.



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 CodePipeline consola, puedes encontrar el nombre del artefacto de salida de cada etapa pasando el ratón por encima del icono de información del diagrama de canalización. Si tu proceso de procesamiento prueba tu aplicación directamente desde la etapa de origen, elige. MyApp Si tu canalización incluye una etapa de compilación, elige MyAppBuild.



15. En la parte inferior del panel, seleccione Añadir acción.
16. En el CodePipeline panel, selecciona Guardar cambio de canalización y, a continuación, selecciona Guardar cambio.
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 Herramientas de AWS para 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 AWS SDK y el proveedor AWS CLI permiten hacerlo.

El AWS SDK proporciona acceso a todos los AWS servicios, incluidos Device Farm, Amazon S3 y más. Para obtener más información, consulte

- las AWS herramientas y SDKs
- la referencia de la API de AWS Device Farm

Ejemplo: usar la AWS CLI o el SDK para cargar una aplicación o una prueba en Device Farm

Los siguientes ejemplos muestran cómo crear una carga en Device Farm mediante la AWS CLI o el AWS SDK en varios idiomas. Las cargas son los elementos fundamentales para programar las pruebas en Device Farm e incluyen lo siguiente:

- Tu aplicación
- ¿Tu prueba
- Su archivo de especificaciones de prueba

Las cargas se crean mediante la `CreateUpload`API. Esta API devuelve una URL prefirmada de S3 a la que puedes enviar la carga mediante una solicitud HTTP PUT. La URL caduca a las 24 horas.

AWS CLI

Nota: en este ejemplo se usa la herramienta de línea de comandos *`curl`* para enviar la aplicación a Device Farm.

En primer lugar, crea un proyecto si aún no lo has hecho.

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

Esto mostrará un resultado como el siguiente:

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

A continuación, haz lo siguiente para crear la carga y subirla a Device Farm. En este ejemplo, crearemos la carga de una aplicación para Android con un archivo APK local. Para obtener más información sobre los tipos de carga, incluidos los detalles sobre los tipos de carga de aplicaciones iOS, consulta nuestra documentación de API para crear una Upload.

```
$ export APP_PATH="/local/path/to/my_sample_app.apk"
$ export APP_TYPE="ANDROID_APP"
```

Primero, creamos la carga en Device Farm:

```
$ aws devicefarm create-upload \
  --project-arn "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE" \
  --name "$(basename "$APP_PATH")" \
  --type "$APP_TYPE"
```

Esto mostrará un resultado como el siguiente:

```
{
    "upload": {
        "arn": "arn:aws:devicefarm:us-
west-2:385076942068:upload:490a6350-0ba3-43e5-83f5-d2896b069a34/a120e848-c57b-4e8d-
a720-d750a0c4d936",
        "name": "my_sample_app.apk",
        "created": 1760747318.266,
        "type": "ANDROID_APP",
        "status": "INITIALIZED",
        "url": "https://prod-us-west-2-uploads.s3.dualstack.us-west-2.amazonaws.com/
arn%3Aaws%3Adevicefarm%3Aus-west-2...",
        "category": "PRIVATE"
    }
}
```

A continuación, realiza una llamada PUT con curl para enviar la aplicación al bucket S3 de Device Farm:

```
$ curl -T "$APP_PATH" "https://prod-us-west-2-uploads.s3.dualstack.us-
west-2.amazonaws.com/arn%3Aaws%3Adevicefarm%3Aus-west-2..."
```

Por último, espera a que la aplicación esté en el estado «correctamente»:

```
$ aws devicefarm get-upload --arn "arn:aws:devicefarm:us-
west-2:385076942068:upload:490a6350-0ba3-43e5-83f5-d2896b069a34/a120e848-c57b-4e8d-
a720-d750a0c4d936"
```

Esto mostrará un resultado como el siguiente:

```
{
    "upload": {
        "arn": "arn:aws:devicefarm:us-
west-2:385076942068:upload:490a6350-0ba3-43e5-83f5-d2896b069a34/a120e848-c57b-4e8d-
a720-d750a0c4d936",
        "name": "my_sample_app.apk",
        "created": 1760747318.266,
        "type": "ANDROID_APP",
        "status": "SUCCEEDED",
        "url": "https://prod-us-west-2-uploads.s3.dualstack.us-west-2.amazonaws.com/
arn%3Aaws%3Adevicefarm%3Aus-west-2...",
        "metadata": "{\"activity_name\":
\"com.amazonaws.devicefarm.android.referenceapp.Activities.MainActivity\",
\"package_name\":\"com.amazonaws.devicefarm.android.referenceapp\",...}",
        "category": "PRIVATE"
    }
}
```

Python

Nota: en este ejemplo se utiliza el *`requests`* paquete de terceros para enviar la aplicación a Device Farm, así como el AWS SDK para Python*`boto3`*.

En primer lugar, crea un proyecto si aún no lo has hecho.

```
import boto3

client = boto3.client("devicefarm", region_name="us-west-2")
```

```
resp = client.create_project(name="MyProjectName")

print(resp)
# Response will be something like:
# {
#     "project": {
#         "name": "MyProjectName",
#         "arn": "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE",
#         "created": 1535675814.414
#     }
# }
```

A continuación, haz lo siguiente para crear la carga y subirla a Device Farm. En este ejemplo, crearemos la carga de una aplicación para Android con un archivo APK local. Para obtener más información sobre los tipos de carga, incluidos los detalles sobre los tipos de carga de aplicaciones iOS, consulta nuestra documentación de API para crear una `Upload`.

```
import os
import time
import datetime
import requests
from pathlib import Path
import boto3


def upload_device_farm_file():
    project_arn = "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE"
    app_path = Path("/local/path/to/my_sample_app.apk")
    file_type = "ANDROID_APP"

    if not app_path.is_file():
        raise RuntimeError(f"{app_path} is not a valid app file path")

    client = boto3.client("devicefarm", region_name="us-west-2")

    # 1) Create the upload in Device Farm
    create = client.create_upload(
        projectArn=project_arn,
        name=app_path.name,
        type=file_type,
        contentType="application/octet-stream",
```

```
    )
    upload = create["upload"]
    upload_arn = upload["arn"]
    upload_url = upload["url"]
    # This will show output such as the following:
    # { "upload": { "arn": "...", "name": "my_sample_app.apk", "type":
 "ANDROID_APP", "status": "INITIALIZED", "url": "https://..." } }

    # 2) Do an HTTP PUT command to push the file to the pre-signed S3 URL
    with app_path.open("rb") as fh:
        print(f"Uploading {app_path.name} to Device Farm...")
        put_resp = requests.put(upload_url, data=fh, headers={"Content-Type":
 "application/octet-stream"})
        put_resp.raise_for_status()

    # 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
    timeout_seconds = 30
    start = time.time()
    while True:
        get_resp = client.get_upload(arn=upload_arn)
        status = get_resp["upload"]["status"]
        msg = get_resp["upload"].get("message") or
 get_resp["upload"].get("metadata") or ""
        elapsed = datetime.timedelta(seconds=int(time.time() - start))
        print(f"[{elapsed}] status={status}{' - ' + msg if msg else ''}")

        if status == "SUCCEEDED":
            print(f"Upload complete: {upload_arn}")
            return upload_arn
        if status == "FAILED":
            raise RuntimeError(f"Device Farm failed to process upload: {msg}")

        if (time.time() - start) > timeout_seconds:
            raise RuntimeError(f"Timed out after {timeout_seconds}s waiting for
 upload to process (last status={status}).")

        time.sleep(1)

upload_device_farm_file()
```

Java

Nota: en este ejemplo se utiliza el AWS SDK para Java v2 y se *`HttpClient`* envía la aplicación a Device Farm, y es compatible con las versiones 11 y superiores de JDK.

En primer lugar, crea un proyecto si aún no lo has hecho.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import software.amazon.awssdk.services.devicefarm.model.CreateProjectRequest;
import software.amazon.awssdk.services.devicefarm.model.CreateProjectResponse;

try (DeviceFarmClient client = DeviceFarmClient.builder()
        .region(Region.US_WEST_2)
        .build()) {
    CreateProjectResponse resp = client.createProject(
        CreateProjectRequest.builder().name("MyProjectName").build());
    System.out.println(resp.project());
    // Response will be something like:
    // Project{name=MyProjectName, arn=arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-..., created=...}
}
```

A continuación, haz lo siguiente para crear la carga y subirla a Device Farm. En este ejemplo, crearemos la carga de una aplicación para Android con un archivo APK local. Para obtener más información sobre los tipos de carga, incluidos los detalles sobre los tipos de carga de aplicaciones iOS, consulta nuestra documentación de API para crear una `Upload`.

```
import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.time.Duration;
import java.time.Instant;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.devicefarm.DeviceFarmClient;
import software.amazon.awssdk.services.devicefarm.model.CreateUploadRequest;
import software.amazon.awssdk.services.devicefarm.model.CreateUploadResponse;
import software.amazon.awssdk.services.devicefarm.model.GetUploadRequest;
import software.amazon.awssdk.services.devicefarm.model.GetUploadResponse;
import software.amazon.awssdk.services.devicefarm.model.Upload;
import software.amazon.awssdk.services.devicefarm.model.UploadType;
```

```
public class DeviceFarmUploader {

    public static String upload(String projectArn, Path appPath) throws Exception {
        if (projectArn == null || projectArn.isEmpty()) {
            throw new IllegalArgumentException("Missing projectArn");
        }
        if (!Files.isRegularFile(appPath)) {
            throw new IllegalArgumentException("Invalid app path: " + appPath);
        }

        String fileName = appPath.getFileName().toString().trim();
        UploadType type = UploadType.ANDROID_APP;

        // Build a reusable HttpClient
        HttpClient http = HttpClient.newBuilder()
                .version(HttpClient.Version.HTTP_1_1)
                .connectTimeout(Duration.ofSeconds(10))
                .build();

        try (DeviceFarmClient client = DeviceFarmClient.builder()
                .region(Region.US_WEST_2)
                .build()) {

            // 1) Create the upload in Device Farm
            CreateUploadResponse create =
 client.createUpload(CreateUploadRequest.builder()
                    .projectArn(projectArn)
                    .name(fileName)
                    .type(type)
                    .contentType("application/octet-stream")
                    .build());

            Upload upload = create.upload();
            String uploadArn = upload.arn();
            String url = upload.url();
            // This will show output such as the following:
            // { "upload": { "arn": "...", "name": "my_sample_app.apk", "type":
 "ANDROID_APP", "status": "INITIALIZED", "url": "https://..." } }

            // 2) PUT file to pre-signed URL using HttpClient
            HttpRequest put = HttpRequest.newBuilder(URI.create(url))
                    .timeout(Duration.ofMinutes(15))
                    .header("Content-Type", "application/octet-stream")
```

```
                    .PUT(HttpRequest.BodyPublishers.ofFile(appPath))
                    .build();

            HttpResponse<Void> resp = http.send(put,
 HttpResponse.BodyHandlers.discarding());
            int code = resp.statusCode();
            if (code / 100 != 2) {
                throw new IOException("Failed PUT to S3 pre-signed URL, HTTP " +
 code);
            }

            // 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
            Instant deadline = Instant.now().plusSeconds(30); // 30-second timeout
            while (true) {
                GetUploadResponse got = client.getUpload(GetUploadRequest.builder()
                        .arn(uploadArn)
                        .build());

                String status = got.upload().statusAsString();
                String msg = got.upload().metadata();
                System.out.println("status=" + status + (msg != null ? " - " + msg :
 ""));

                if ("SUCCEEDED".equals(status)) return uploadArn;
                if ("FAILED".equals(status)) throw new RuntimeException("Upload
 failed: " + msg);
                if (Instant.now().isAfter(deadline)) {
                    throw new RuntimeException("Timeout waiting for processing, last
 status=" + status);
                }
                Thread.sleep(2000);
            }
        }
    }

    public static void main(String[] args) throws Exception {
        String projectArn = "arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-c861-4c0a-b1d5-12345EXAMPLE";
        Path appPath = Paths.get("/local/path/to/my_sample_app.apk");
        String result = upload(projectArn, appPath);
        System.out.println("Upload ARN: " + result);
    }
}
```

JavaScript

Nota: en este ejemplo se usa el AWS SDK para JavaScript (v3) y Node 18+ *fetch* para enviar la aplicación a Device Farm.

En primer lugar, crea un proyecto si aún no lo has hecho.

```
import { DeviceFarmClient, CreateProjectCommand } from "@aws-sdk/client-device-
farm";

const df = new DeviceFarmClient({ region: "us-west-2" });
const resp = await df.send(new CreateProjectCommand({ name: "MyProjectName" }));
console.log(resp);
// Response will be something like:
// { project: { name: 'MyProjectName', arn: 'arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-...', created: 1535675814.414 } }
```

A continuación, haz lo siguiente para crear la carga y subirla a Device Farm. En este ejemplo, crearemos la carga de una aplicación para Android con un archivo APK local. Para obtener más información sobre los tipos de carga, incluidos los detalles sobre los tipos de carga de aplicaciones iOS, consulta nuestra documentación de API para crear una `Upload`.

```
import { DeviceFarmClient, CreateUploadCommand, GetUploadCommand } from "@aws-sdk/
client-device-farm";
import { createReadStream } from "fs";
import { basename } from "path";

const projectArn = "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-
c861-4c0a-b1d5-12345EXAMPLE";
const appPath = "/local/path/to/my_sample_app.apk";
const name = basename(appPath).trim();
const type = "ANDROID_APP";

const client = new DeviceFarmClient({ region: "us-west-2" });

// 1) Create the upload in Device Farm
const create = await client.send(new CreateUploadCommand({
  projectArn,
  name,
  type,
  contentType: "application/octet-stream",
}));
```

```
const uploadArn = create.upload.arn;
const url = create.upload.url;
// This will show output such as the following:
// { upload: { arn: '...', name: 'my_sample_app.apk', type: 'ANDROID_APP', status:
 'INITIALIZED', url: 'https://...' } }

// 2) PUT to pre-signed URL
const putResp = await fetch(url, {
  method: "PUT",
  headers: { "Content-Type": "application/octet-stream" },
  body: createReadStream(appPath),
});
if (!putResp.ok) {
  throw new Error(`Failed PUT to pre-signed URL: ${putResp.status} ${await
 putResp.text().catch(()=>"")}`);
}

// 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
const deadline = Date.now() + (30 * 1000); // 30-second timeout
while (true) {
  const response = await client.send(new GetUploadCommand({ arn: uploadArn }));
  const { status, message, metadata } = response.upload;
  console.log(`status=${status}${message ? " - " + message : metadata ? " - " +
 metadata : ""}`);
  if (status === "SUCCEEDED") {
    console.log("Upload complete:", uploadArn);
    break;
  }
  if (status === "FAILED") {
    throw new Error(`Upload failed: ${message || metadata || "unknown"}`);
  }
  if (Date.now() > deadline) throw new Error(`Timeout waiting for processing (last
 status=${status})`);
  await new Promise(r => setTimeout(r, 2000));
}
```

C#

Nota: en este ejemplo se usa el AWS SDK para .NET y *`HttpClient`* se envía la aplicación a Device Farm.

En primer lugar, crea un proyecto si aún no lo has hecho.

```
using System;
```

```
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

using var client = new AmazonDeviceFarmClient(RegionEndpoint.USWest2);
var resp = await client.CreateProjectAsync(new CreateProjectRequest { Name =
 "MyProjectName" });
Console.WriteLine(resp.Project);
// Response will be something like:
// { Name = MyProjectName, Arn = arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-..., Created = ... }
```

A continuación, haz lo siguiente para crear la carga y subirla a Device Farm. En este ejemplo, crearemos la carga de una aplicación para Android con un archivo APK local. Para obtener más información sobre los tipos de carga, incluidos los detalles sobre los tipos de carga de aplicaciones iOS, consulta nuestra documentación de API para crear una `Upload`.

```
using System;
using System.IO;
using System.Net.Http;
using System.Threading.Tasks;
using System.Net.Http.Headers;
using Amazon;
using Amazon.DeviceFarm;
using Amazon.DeviceFarm.Model;

class DeviceFarmUploader
{
    public static async Task<string> UploadAsync(string projectArn, string appPath)
    {
        if (string.IsNullOrWhiteSpace(projectArn)) throw new
 ArgumentException("Missing projectArn");
        if (!File.Exists(appPath)) throw new ArgumentException($"Invalid app path:
 {appPath}");
        var type = UploadType.ANDROID_APP;

        using var client = new AmazonDeviceFarmClient(RegionEndpoint.USWest2);
        // 1) Create the upload in Device Farm
        var create = await client.CreateUploadAsync(new CreateUploadRequest
        {
            ProjectArn = projectArn,
            Name = Path.GetFileName(appPath),
            Type = type,
```

```
            ContentType = "application/octet-stream"
        });

        var uploadArn = create.Upload.Arn;
        var url = create.Upload.Url;
        // This will show output such as the following:
        // { Upload: { Arn = ..., Name = my_sample_app.apk, Type = ANDROID_APP,
Status = INITIALIZED, Url = https://... } }

        // 2) PUT file to pre-signed URL
        using (var http = new HttpClient())
        using (var fs = File.OpenRead(appPath))
        using (var content = new StreamContent(fs))
        {
            content.Headers.Add("Content-Type", "application/octet-stream");
            var resp = await http.PutAsync(url, content);
            if (!resp.IsSuccessStatusCode)
                throw new Exception($"Failed PUT to pre-signed URL:
{(int)resp.StatusCode} {await resp.Content.ReadAsStringAsync()}");
        }

        // 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
        var deadline = DateTime.UtcNow.AddSeconds(30); // 30-second timeout
        while (true)
        {
            var got = await client.GetUploadAsync(new GetUploadRequest { Arn =
uploadArn });
            var status = got.Upload.Status.Value;
            var msg = got.Upload.Message ?? got.Upload.Metadata;
            Console.WriteLine($"status={status}{(string.IsNullOrEmpty(msg) ? "" : "
- " + msg)}");

            if (status == UploadStatus.SUCCEEDED.Value) return uploadArn;
            if (status == UploadStatus.FAILED.Value) throw new Exception($"Upload
failed: {msg}");
            if (DateTime.UtcNow > deadline) throw new TimeoutException($"Timeout
waiting for processing (last status={status})");
            await Task.Delay(2000);
        }
    }

    static async Task Main()
    {
```

```
        var projectArn = "arn:aws:devicefarm:us-
west-2:123456789101:project:5e01a8c7-c861-4c0a-b1d5-12345EXAMPLE";
        var appPath = "/local/path/to/my_sample_app.apk";
        var result = await UploadAsync(projectArn!, appPath!);
        Console.WriteLine("Upload ARN: " + result);
    }
}
```

Ruby

Nota: en este ejemplo se usa el AWS SDK para Ruby y *`Net::HTTP`* se envía la aplicación a Device Farm.

En primer lugar, crea un proyecto si aún no lo has hecho.

```
require "aws-sdk-devicefarm"

client = Aws::DeviceFarm::Client.new(region: "us-west-2")
resp = client.create_project(name: "MyProjectName")
puts resp.project.inspect
# Response will be something like:
# #<struct Aws::DeviceFarm::Types::Project name="MyProjectName",
 arn="arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-...",
 created=1535675814.414>
```

A continuación, haz lo siguiente para crear la carga y subirla a Device Farm. En este ejemplo, crearemos la carga de una aplicación para Android con un archivo APK local. Para obtener más información sobre los tipos de carga, incluidos los detalles sobre los tipos de carga de aplicaciones iOS, consulta nuestra documentación de API para crear una `Upload`.

```
require "aws-sdk-devicefarm"
require "net/http"
require "uri"

project_arn = "arn:aws:devicefarm:us-west-2:123456789101:project:5e01a8c7-c861-4c0a-
b1d5-12345EXAMPLE"
app_path    = "/local/path/to/my_sample_app.apk"
raise "Invalid APP_PATH: #{app_path}" unless File.file?(app_path)
type = "ANDROID_APP"

client = Aws::DeviceFarm::Client.new(region: "us-west-2")

# 1) Create the upload in Device Farm
```

```
create = client.create_upload(
  project_arn: project_arn,
  name: File.basename(app_path),
  type: type,
  content_type: "application/octet-stream"
)

upload_arn = create.upload.arn
url = create.upload.url
# This will show output such as the following:
# #<Upload arn="...", name="my_sample_app.apk", type="ANDROID_APP",
 status="INITIALIZED", url="https://...">

# 2) PUT the file to the pre-signed URL
uri = URI.parse(url)
Net::HTTP.start(uri.host, uri.port, use_ssl: (uri.scheme == "https")) do |http|
  req = Net::HTTP::Put.new(uri)
  req["Content-Type"] = "application/octet-stream"
  req.body_stream = File.open(app_path, "rb")
  req.content_length = File.size(app_path)
  resp = http.request(req)
  raise "Failed PUT: #{resp.code} #{resp.body}" unless resp.code.to_i / 100 == 2
end

# 3) Wait for the app to be in "SUCCEEDED" status (or fail/timeout)
deadline = Time.now + 30 # 30-second timeout
loop do
  got = client.get_upload(arn: upload_arn)
  status = got.upload.status
  msg = got.upload.message || got.upload.metadata
  puts "status=#{status}#{msg ? " - #{msg}" : ""}"

  case status
  when "SUCCEEDED" then puts "Upload complete: #{upload_arn}"; break
  when "FAILED"    then raise "Upload failed: #{msg}"
  end
  raise "Timeout waiting for processing (last status=#{status})" if Time.now >
 deadline
  sleep 2
end
```

Ejemplo: usar el AWS SDK para iniciar una ejecución de Device Farm y recolectar artefactos

El siguiente ejemplo proporciona una beginning-to-end demostración de cómo se puede utilizar el AWS SDK 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.

> **Note**
>
> Para solucionar problemas de pruebas de Appium que fallan inesperadamente en Device Farm, consulta nuestra guía de pruebas de Appium del lado del cliente

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 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 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 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 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**
>
> No hemos podido encontrar el AndroidManifest archivo.xml válido en su aplicación. 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 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 Pruebas de Android en AWS Device Farm.

Algunas ventanas de mi aplicación de Android se muestran en blanco o en negro

Si está probando una aplicación de Android y observa que algunas ventanas de la aplicación aparecen en negro en la grabación de video 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, es posible que la característica de grabación de pantalla de Device Farm (tanto para las pruebas de automatización como para las de acceso remoto) muestre una pantalla negra en lugar de la ventana de la aplicación, si esta incorpora este indicador.

Los desarrolladores suelen utilizar este indicador 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, hable con los desarrolladores para obtener una versión de la aplicación para las pruebas que no incorpore este indicador.

Además, Device Farm puede interactuar con las ventanas de aplicaciones que tienen este indicador. Por lo tanto, aunque la página de inicio de sesión de su aplicación aparezca en negro, es posible que pueda introducir sus credenciales de registro en la aplicación (y así ver las páginas no bloqueadas por el indicador `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_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 Ejecute automáticamente pruebas de Appium en 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 Ejecute automáticamente pruebas de Appium en 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 Ejecute automáticamente pruebas de Appium en 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 Ejecute automáticamente pruebas de Appium en 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 Ejecute automáticamente pruebas de Appium en 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 Ejecute automáticamente pruebas de Appium en 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 Ejecute automáticamente pruebas de Appium en Device Farm.

Solución de problemas de las pruebas de aplicaciones JUnit web Java de Appium en AWS Device Farm

En el siguiente tema se enumeran los mensajes de error que se producen al cargar las pruebas de las aplicaciones JUnit web Java de Appium y se recomiendan soluciones alternativas para resolver cada error. Para obtener más información acerca del uso de Appium con Device Farm, consulte the section called “Pruebas automáticas de 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. zip-with-dependencies

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 JUnit paquete Java de Appium válido debería producir un resultado como el siguiente:

```
.
|— 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. zip-with-dependencies

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 JUnit paquete Java de Appium es válido, encontrará el *dependency-jars* directorio 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. zip-with-dependencies

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 JUnit paquete Java de Appium es válido, encontrará al menos un *jar* archivo 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. zip-with-dependencies

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 JUnit paquete Java de Appium es válido, encontrará al menos un *jar* archivo como *acme-android-appium-1.0-SNAPSHOT-tests.jar* en nuestro ejemplo. El nombre del archivo puede ser diferente, pero debe 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 .
```

Deberías encontrar al menos un archivo jar como *acme-android-appium-1.0-SNAPSHOT-tests.jar* en nuestro ejemplo. El nombre del archivo puede ser diferente, pero debe 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**
>
> No hemos podido encontrar un valor de JUnit versión. Descomprima el paquete de prueba y abra el directorio dependency-jars, compruebe que el archivo JUnit JAR está dentro del directorio e inténtelo de nuevo.

En el siguiente ejemplo, el nombre del paquete es .zip. zip-with-dependencies

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 JUnit paquete Java de Appium es válido, encontrará el archivo de JUnit dependencias similar al archivo jar de nuestro ejemplo*junit-4.10.jar*. El nombre debe estar formado 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**
>
> Descubrimos que la JUnit versión era inferior a la versión mínima 4.10 que admitimos. Cambia la JUnit versión e inténtalo de nuevo.

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 .
   ```

   Deberías encontrar un archivo de JUnit dependencias como el *junit-4.10.jar* de 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 sus pruebas no se ejecuten correctamente si la JUnit versión especificada en su paquete de prueba es inferior a la versión 4.10 mínima que admitimos.

Para obtener más información, consulte Ejecute automáticamente pruebas de Appium en 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 Ejecute automáticamente pruebas de Appium en 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 Ejecute automáticamente pruebas de Appium en 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 Ejecute automáticamente pruebas de Appium en 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 Ejecute automáticamente pruebas de Appium en 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 Ejecute automáticamente pruebas de Appium en 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 JUnit paquete Java de Appium válido debería producir un resultado como el siguiente:

   ```
   .
   |— 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 Ejecute automáticamente pruebas de Appium en 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. zip-with-dependencies

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 JUnit paquete Java de Appium es válido, encontrará el *`dependency-jars`* directorio 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 Ejecute automáticamente pruebas de Appium en Device Farm.

APPIUM_WEB_JAVA_TESTNG_TEST_PACKAGE_JAR_MISSING_IN_DEPEND

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. zip-with-dependencies

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 JUnit paquete Java de Appium es válido, encontrará al menos un *jar* archivo 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 Ejecute automáticamente pruebas de Appium en 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. zip-with-dependencies

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 JUnit paquete Java de Appium es válido, encontrará al menos un *jar* archivo como *acme-android-appium-1.0-SNAPSHOT-tests.jar* en nuestro ejemplo. El nombre del archivo puede ser diferente, pero debe 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 Ejecute automáticamente pruebas de Appium en 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 .
   ```

Deberías encontrar al menos un archivo jar como *acme-android-appium-1.0-SNAPSHOT-tests.jar* en nuestro ejemplo. El nombre del archivo puede ser diferente, pero debe 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 Ejecute automáticamente pruebas de Appium en 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 Ejecute automáticamente pruebas de Appium en 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 Ejecute automáticamente pruebas de Appium en 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 Ejecute automáticamente pruebas de Appium en 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 Ejecute automáticamente pruebas de Appium en 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 Ejecute automáticamente pruebas de Appium en 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 Ejecute automáticamente pruebas de Appium en 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 Ejecute automáticamente pruebas de Appium en 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 Ejecute automáticamente pruebas de Appium en 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 Ejecute automáticamente pruebas de Appium en Device Farm.

APPIUM_PYTHON_TEST_PACKAGE_DEPENDENCY_WHEELS_INSUFFICIEN

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 Ejecute automáticamente pruebas de Appium en 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 Ejecute automáticamente pruebas de Appium en 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 Python de Appium es válido, encontrará al menos un archivo *.whl* dependiente, como los archivos resaltados, dentro del *wheelhouse* directorio.

   ```
   .
   |-- 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 Ejecute automáticamente pruebas de Appium en 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 Python de Appium es válido, encontrará al menos un archivo *.whl* dependiente, como los archivos resaltados, dentro del *wheelhouse* directorio. El nombre del archivo puede ser diferente, pero debe 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 Ejecute automáticamente pruebas de Appium en 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 Python de Appium es válido, encontrará el *tests* directorio 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 Ejecute automáticamente pruebas de Appium en 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 Python de Appium es válido, encontrará el *tests* directorio dentro del directorio de trabajo. El nombre del archivo puede ser diferente, pero debe empezar *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 Ejecute automáticamente pruebas de Appium en 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 Python de Appium es válido, encontrará el *requirements.txt* archivo 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 Ejecute automáticamente pruebas de Appium en 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 .
```

Deberías encontrar el *requirement.txt* archivo 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 Ejecute automáticamente pruebas de Appium en 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 Ejecute automáticamente pruebas de Appium en 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 Ejecute automáticamente pruebas de Appium en 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.

> **Note**
>
> Para conocer las consideraciones importantes a la hora de utilizar las pruebas de instrumentación en AWS Device Farm, consulte Instrumentación para Android y AWS Device Farm.

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 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.

```
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 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.

```
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 de la : `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 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.

```
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 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.

```
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 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 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 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 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 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 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 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 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 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 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 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 Integración de Device Farm con XCTest iOS.

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 Integración de Device Farm con XCTest iOS.

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 Integración de Device Farm con XCTest iOS.

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 Integración de Device Farm con XCTest iOS.

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 Integración de Device Farm con XCTest iOS.

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.

```
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 Integración de la XCTest interfaz de usuario para iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

`We could not find the Payload directory inside your test package. Please unzip your test package, verify that the Payload directory is inside the package, and try again.`

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 Integración de la XCTest interfaz de usuario para iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_APP_DIR_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

```
We could not find the .app directory inside the Payload directory. Please unzip your test package and then open the Payload directory, verify that the .app directory is inside the directory, and try again.
```

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 Integración de la XCTest interfaz de usuario para iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_PLUGINS_DIR_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

```
We could not find the Plugins directory inside the .app directory. Please unzip your test package and then open the .app directory, verify that the Plugins directory is inside the directory, and try again.
```

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 Integración de la XCTest interfaz de usuario para iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_XCTEST_DIR_MISSING_IN_PLUGINS_DIR

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

```
We could not find the .xctest directory inside the plugins directory.
Please unzip your test package and then open the plugins directory, verify
that the .xctest directory is inside the directory, and try again.
```

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 Integración de la XCTest interfaz de usuario para iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

`We could not find the Info.plist file inside the .app directory. Please unzip your test package and then open the .app directory, verify that the Info.plist file is inside the directory, and try again.`

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 Integración de la XCTest interfaz de usuario para iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_PLIST_FILE_MISSING_IN_XCTEST_DIR

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

```
We could not find the Info.plist file inside the .xctest directory. Please
unzip your test package and then open the .xctest directory, verify that
the Info.plist file is inside the directory, and try again.
```

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 Integración de la XCTest interfaz de usuario para iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_CPU_ARCHITECTURE_VALUE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

```
We could not the CPU architecture value in the Info.plist file. Please
unzip your test package and then open the Info.plist file inside the .app
```

```
directory, verify that the key "UIRequiredDeviceCapabilities" is specified,
and try again.
```

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 Integración de la XCTest interfaz de usuario para iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_PLATFORM_VALUE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

```
We could not find the platform value in the Info.plist. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the key "CFBundleSupportedPlatforms" is specified, and try again.
```

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 Integración de la XCTest interfaz de usuario para iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_WRONG_PLATFORM_DEVICE_VALUE

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

```
We found the platform device value was wrong in the Info.plist file. Please
unzip your test package and then open the Info.plist file inside the .app
directory, verify that the value of the key "CFBundleSupportedPlatforms"
does not contain the keyword "simulator", and try again.
```

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 Integración de la XCTest interfaz de usuario para iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_FORM_FACTOR_VALUE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

```
We could not the form factor value in the Info.plist. Please unzip your
test package and then open the Info.plist file inside the .app directory,
verify that the key "UIDeviceFamily" is specified, and try again.
```

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 Integración de la XCTest interfaz de usuario para iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_PACKAGE_NAME_VALUE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

`We could not find the package name value in the Info.plist file. Please unzip your test package and then open the Info.plist file inside the .app directory, verify that the key "CFBundleIdentifier" is specified, and try again.`

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 Integración de la XCTest interfaz de usuario para iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_EXECUTABLE_VALUE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

```
We could not find the executable value in the Info.plist file. Please
unzip your test package and then open the Info.plist file inside the .app
directory, verify that the key "CFBundleExecutable" is specified, and try
again.
```

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 Integración de la XCTest interfaz de usuario para iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_TEST_PACKAGE_NAME_VALUE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

```
We could not find the package name value in the Info.plist file inside
the .xctest directory. Please unzip your test package and then open
the Info.plist file inside the .xctest directory, verify that the key
"CFBundleIdentifier" is specified, and try again.
```

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 Integración de la XCTest interfaz de usuario para iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_TEST_EXECUTABLE_VALUE_MISSING

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

```
We could not find the executable value in the Info.plist file inside
the .xctest directory. Please unzip your test package and then open
the Info.plist file inside the .xctest directory, verify that the key
"CFBundleExecutable" is specified, and try again.
```

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 Integración de la XCTest interfaz de usuario para iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_MULTIPLE_APP_DIRS

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

`We found multiple .app directories inside your test package. Please unzip your test package, verify that only a single .app directory is present inside the package, then try again.`

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip swift-sample-UI.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 UI es válido, solo debería encontrar un directorio `.app`, como `swift-sampleUITests-Runner.app`, en nuestro ejemplo dentro del paquete de prueba .zip.

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

Para obtener más información, consulte Integración de la XCTest interfaz de usuario para iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_MULTIPLE_IPA_DIRS

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

`We found multiple .ipa directories inside your test package. Please unzip your test package, verify that only a single .ipa directory is present inside the package, then try again.`

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

```
$ unzip swift-sample-UI.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 UI es válido, solo debería encontrar un directorio `.ipa`, como `sampleUITests.ipa`, en nuestro ejemplo dentro del paquete de prueba .zip.

```
.
`--swift-sample-UI.zip--(directory)
    `-- sampleUITests.ipa (directory)
            `-- Payload (directory)
                `-- swift-sampleUITests-Runner.app (directory)
    `-- (any other files)
```

Para obtener más información, consulte Integración de la XCTest interfaz de usuario para iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_BOTH_APP_AND_IPA_DIR_PRESENT

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

`We found both .app and .ipa files inside your test package. Please unzip your test package, verify that only a single .app or .ipa file is present inside the package, then try again.`

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip swift-sample-UI.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 UI es válido, debería encontrar un directorio `.ipa`, como `sampleUITests.ipa`, o un directorio `.app`, como `swift-sampleUITests-Runner.app`, en nuestro ejemplo dentro del paquete de prueba .zip. Puede consultar un ejemplo de paquete de prueba XCTEST_UI válido en nuestra documentación sobre Integración de la XCTest interfaz de usuario para iOS con Device Farm.

   ```
   .
   ```

```
`--swift-sample-UI.zip--(directory)
    `-- sampleUITests.ipa (directory)
            `-- Payload (directory)
                `-- swift-sampleUITests-Runner.app (directory)
   `-- (any other files)
```

o

```
.
`--swift-sample-UI.zip--(directory)
    `-- swift-sampleUITests-Runner.app (directory)
            |-- Info.plist
            |-- Plugins (directory)
            `-- (any other files)
   `-- (any other files)
```

Para obtener más información, consulte Integración de la XCTest interfaz de usuario para iOS con Device Farm.

XCTEST_UI_TEST_PACKAGE_PAYLOAD_DIR_PRESENT_IN_ZIP

Si aparece el siguiente mensaje, siga estos pasos para solucionar el problema.

```
We found a Payload directory inside your .zip test package. Please unzip your test package, ensure that a Payload directory is not present in the package, then try again.
```

1. Copie el paquete de pruebas a su directorio de trabajo y, a continuación, ejecute el siguiente comando:

   ```
   $ unzip swift-sample-UI.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 UI es válido, no debería encontrar ningún directorio de carga útil dentro de su paquete de prueba.

```
.
`--swift-sample-UI.zip--(directory)
    `-- swift-sampleUITests-Runner.app (directory)
            |-- Info.plist
            |-- Plugins (directory)
            `-- (any other files)
   `-- Payload (directory) [This directory should not be present]
            |-- (any other files)
   `-- (any other files)
```

Para obtener más información, consulte Integración de la XCTest interfaz de usuario para iOS con Device Farm.

Seguridad en AWS Device Farm

La seguridad en la nube AWS es la máxima prioridad. Como AWS cliente, usted se beneficia de una arquitectura de centro de datos y red diseñada para cumplir con los requisitos de las organizaciones más sensibles a la seguridad.

La seguridad es una responsabilidad compartida entre usted 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 AWS los servicios en la AWS nube. AWS también le proporciona servicios que puede utilizar de forma segura. Los auditores externos prueban y verifican periódicamente la eficacia de nuestra seguridad como parte de los AWS programas de de . Para obtener más información sobre los programas de conformidad aplicables AWS Device Farm, consulte Servicios de AWS dentro del alcance por programa de conformidad .
- Seguridad en la nube: su responsabilidad viene determinada por 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 la normativa 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 utilizas AWS Identity and Access Management (IAM) varía según tu función:

- Usuario del servicio: solicite permisos al administrador si no puede acceder a las características (consulte Solución de problemas de identidad y acceso a AWS Device Farm).
- Administrador del servicio: determine el acceso de los usuarios y envíe las solicitudes de permiso (consulte Cómo funciona AWS Device Farm con IAM).
- Administrador de IAM: escribe las políticas para administrar el acceso (consulte Ejemplos de políticas basadas en identidad de AWS Device Farm).

Autenticación con identidades

La autenticación es la forma en que inicias sesión AWS con tus credenciales de identidad. Debe autenticarse como usuario de Usuario raíz de la cuenta de AWS IAM o asumir una función de IAM.

Puede iniciar sesión como una identidad federada con las credenciales de una fuente de identidad, como AWS IAM Identity Center (IAM Identity Center), la autenticación de inicio de sesión único o las credenciales. Google/Facebook Para obtener más información sobre el inicio de sesión, consulte Cómo iniciar sesión en la Cuenta de AWS en la Guía del usuario de AWS Sign-In .

Para el acceso programático, AWS proporciona un SDK y una CLI para firmar criptográficamente las solicitudes. Para obtener más información, consulte AWS Signature Version 4 para solicitudes de API en la Guía del usuario de IAM.

Cuenta de AWS usuario root

Al crear un Cuenta de AWS, se comienza con una identidad de inicio de sesión denominada usuario Cuenta de AWS raíz que tiene acceso completo a todos Servicios de AWS los recursos. Se recomiendaencarecidamente que no utilice el usuario raíz para las tareas diarias. Para ver la lista completa de las tareas que requieren credenciales de 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 con permisos específicos para una sola persona o aplicación. Recomendamos el uso de credenciales temporales en lugar de usuarios de IAM con credenciales de

larga duración. Para obtener más información, consulte Exigir a los usuarios humanos que utilicen la federación con un proveedor de identidad para acceder AWS mediante credenciales temporales en la Guía del usuario de IAM.

Un grupo de IAM especifica un conjunto de usuarios de IAM y facilita la administración de los permisos para grupos grandes de usuarios. Para obtener más información, consulte Casos de uso para usuarios de IAM en la Guía del usuario de IAM.

Roles de IAM

Un Rol de IAM es una identidad con permisos específicos que proporciona credenciales temporales. Puede asumir un rol cambiando de un rol de usuario a uno de IAM (consola) o llamando a una AWS CLI operación de AWS API. Para obtener más información, consulte Métodos para asumir un rol en la Guía del usuario de IAM.

Los roles de IAM son útiles para el acceso de usuario federado, los permisos de usuario de IAM temporales, el acceso entre cuentas, el acceso entre servicios y las aplicaciones que se ejecutan en Amazon EC2. Para obtener más información, consulte Acceso a recursos entre cuentas en IAM 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 visión general de cómo funcionan Device Farm y otros AWS servicios con IAM, consulte AWS Servicios 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 usar las políticas de AWS JSON 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. 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 usar las políticas de AWS JSON 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. Como práctica recomendada, especifique un recurso utilizando el Nombre de recurso de Amazon (ARN). En el caso de las acciones que no admiten permisos por recurso, 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 sobre el formato de ARNs, consulte Amazon Resource Names (ARNs) y AWS Service Namespaces.

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 sola sentencia, sepárelos ARNs con comas.

```
"Resource": [
      "resource1",
      "resource2"
```

Para ver una lista de los tipos de recursos de Device Farm y sus tipos ARNs, consulte los tipos de recursos definidos AWS Device Farm en la Referencia de autorización de servicios 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 usar las políticas de AWS JSON 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` especifica cuándo se ejecutan las instrucciones en función de criterios definidos. Puede crear expresiones condicionales que utilizan operadores de condición, tales como igual o menor que, para que la condición de la política coincida con los valores de la solicitud. Para ver todas las claves de condición AWS globales, consulte las claves de contexto de condición AWS globales 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 AWS globales, consulte las claves de contexto de condición AWS globales 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 listas de control de acceso (ACLs).

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 tu AWS cuenta que tiene 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. Para obtener credenciales de seguridad temporales, puede llamar a operaciones de AWS STS API como AssumeRoleo GetFederationToken.

Roles vinculados a servicios

Los roles vinculados a un servicio permiten a AWS los servicios acceder a los recursos de otros servicios para completar una acción en tu 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 del acceso con políticas

El acceso se controla AWS creando políticas y adjuntándolas a AWS identidades o recursos. Una política define los permisos cuando están asociados a una identidad o un recurso. AWS evalúa estas políticas cuando un director hace una solicitud. La mayoría de las políticas se almacenan AWS como documentos JSON. Para obtener más información sobre los documentos de políticas de JSON, consulte Información general de políticas de JSON en la Guía del usuario de IAM.

Mediante las políticas, los administradores especifican quién tiene acceso a qué, definiendo qué entidad principal puede realizar acciones sobre qué recursos y en qué condiciones.

De forma predeterminada, los usuarios y los roles no tienen permisos. Un administrador de IAM crea políticas de IAM y las agrega a roles, que los usuarios pueden asumir posteriormente. Las políticas de IAM definen permisos independientemente del método que se utilice para realizar la operación.

Políticas basadas en identidades

Las políticas basadas en identidad son documentos de política de permisos JSON que asocia a una identidad (usuario, grupo o rol). Estas políticas controlan qué acciones pueden realizar las identidades, en qué recursos y en qué condiciones. Para obtener más información sobre cómo crear una política basada en la identidad, consulte Definición de permisos de IAM personalizados con políticas administradas por el cliente en la Guía del usuario de IAM.

Las políticas basadas en identidad pueden ser políticas insertadas (incrustadas directamente en una sola identidad) o políticas administradas (políticas independientes asociadas a varias identidades). Para obtener información sobre cómo elegir entre políticas administradas e insertadas, consulte Selección 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

Cambio	Descripción	Fecha
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 tipos de políticas adicionales que pueden establecer los permisos máximos que conceden los tipos de políticas más comunes:

- Límites de permisos: establecen los permisos máximos que una política basada en identidad puede conceder a una entidad de IAM. Para obtener más información, consulte Límites de permisos para las entidades de IAM en la Guía del usuario de IAM.
- Políticas de control de servicios (SCPs): especifican los permisos máximos para una organización o unidad organizativa en AWS Organizations. Para obtener más información, consulte Políticas de control de servicios en la Guía del usuario de AWS Organizations .
- Políticas de control de recursos (RCPs): establece los permisos máximos disponibles para los recursos de tus cuentas. Para obtener más información, consulte Políticas de control de recursos (RCPs) en la Guía del AWS Organizations usuario.
- Políticas de sesión: políticas avanzadas que se pasan como parámetro cuando se crea una sesión temporal para un rol o un usuario federado. Para obtener 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 saber cómo se AWS determina si se debe permitir una solicitud cuando se trata de varios tipos de políticas, consulte la 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 con la Consola de administración de AWS AWS CLI, o la AWS API. 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 costos adicionales para su Cuenta de AWS. Siga estas directrices y recomendaciones al crear o editar políticas basadas en identidades:

- Comience con las políticas AWS administradas y avance hacia los permisos con privilegios mínimos: para empezar a conceder permisos a sus usuarios y cargas de trabajo, utilice las políticas AWS administradas que otorgan permisos en muchos casos de uso comunes. Están disponibles en su. Cuenta de AWS Le recomendamos que reduzca aún más los permisos definiendo políticas administradas por el AWS cliente que sean específicas para sus casos de uso. Con el fin de obtener más información, consulte las políticas administradas por AWS o las políticas administradas por AWS para funciones de tarea en la Guía de 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 puedes usar condiciones para conceder el acceso a las acciones del servicio si se utilizan a través de una acción específica Servicio de AWS, por ejemplo CloudFormation. Para obtener más información, consulte Elementos de la política de JSON de IAM: Condición en la Guía del usuario de IAM.

- Utiliza 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 Validación de políticas con el Analizador de acceso de IAM en la Guía del usuario de IAM.
- Requerir autenticación multifactor (MFA): si tiene un escenario que requiere usuarios de IAM o un usuario raíz en Cuenta de AWS su cuenta, active la MFA para mayor seguridad. Para exigir la MFA cuando se invoquen las operaciones de la API, añada condiciones de MFA a sus políticas. Para más información, consulte Acceso seguro a la API con MFA en la Guía del usuario de IAM.

Para obtener más información sobre las prácticas recomendadas de IAM, consulte 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 completar esta acción en la consola o mediante programación mediante la API o. AWS CLI 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 conceder a un usuario de IAM de su AWS cuenta acceso a uno de sus proyectos de prueba del 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 conexión `devicefarm:GetTestGridProject`, la cuenta debe tener los puntos de enlace `devicefarm:ListTestGridSessions`, `devicefarm:GetTestGridSession`, `devicefarm:ListTestGridSessionActions` y `devicefarm:ListTestGridSessionArtifacts`.

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:

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.

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 el Consola de administración de AWS que se indica que no está autorizado a realizar una acción, debe ponerse en contacto con el administrador para obtener 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 lo siguiente: 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.

Algunas Servicios de AWS permiten transferir una función existente a ese servicio en lugar de crear una nueva función de servicio o una función vinculada a un servicio. Para ello, debe tener permisos para transferir la función 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 otorguen 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 AWS administrador. 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. De este modo, podrías dar a alguien acceso permanente a tu 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 otras personas accedan a Device Farm, debe conceder permiso a las personas o aplicaciones que necesiten acceder. Si usa AWS IAM Identity Center para administrar las personas y las aplicaciones, debe asignar conjuntos de permisos a los usuarios o grupos para definir su nivel de acceso. Los conjuntos de permisos crean políticas de IAM y las asignan a los roles de IAM asociados a la persona o aplicación de forma automática. Para obtener más información, consulte la sección Conjuntos de permisos en la Guía del usuario de AWS IAM Identity Center .

Si no utiliza IAM Identity Center, debe crear entidades de IAM (usuarios o roles) para las personas o aplicaciones que necesitan acceso. A continuación, debe asociar una política a la entidad que le conceda los permisos correctos en Device Farm. Una vez concedidos los permisos, proporcione las credenciales al usuario o al desarrollador de la aplicación. Utilizarán esas credenciales para acceder a AWS. Para obtener más información sobre la creación de usuarios, grupos, políticas y permisos de IAM, consulte Identidades de IAM y Políticas y permisos en IAM en la Guía del usuario de IAM.

Quiero permitir que personas ajenas a mi AWS cuenta accedan a mis recursos de Device Farm

Se puede crear un rol que los usuarios de otras cuentas o las personas externas a la organización puedan utilizar para acceder a sus recursos. Se puede especificar una persona de confianza para que asuma el rol. En el caso de los servicios que admiten políticas basadas en recursos o listas de control de acceso (ACLs), puede usar esas políticas para permitir que las personas accedan a sus recursos.

Para obtener 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 sobre cómo proporcionar acceso a los recursos de su Cuentas de AWS propiedad, consulte Proporcionar acceso a un usuario de IAM en otro de su propiedad en la Cuenta de AWS Guía del usuario de IAM.
- Para obtener información sobre cómo proporcionar acceso a tus recursos a terceros Cuentas de AWS, consulta Cómo proporcionar acceso a recursos que Cuentas de AWS 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 (identidad federada) en la Guía del usuario de IAM.
- Para conocer sobre la diferencia entre las políticas basadas en roles y en recursos para el acceso entre cuentas, consulte Acceso a recursos entre cuentas en IAM en la Guía del usuario de IAM.

Validación de 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.

Puedes 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 CSPM: este servicio de AWS proporciona una vista integral de su estado de seguridad en AWS que lo ayuda a verificar si cumple con los estándares y las buenas prácticas de seguridad de la industria.

Protección de datos en AWS Device Farm

El modelo de responsabilidad AWS compartida modelo 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 en la que se ejecutan todos los Nube de AWS. Eres responsable de mantener el control sobre el contenido alojado en esta infraestructura. También eres 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, consulta la publicación de blog sobre el Modelo de responsabilidad compartida de AWS y RGPD en el Blog de seguridad de AWS .

Con fines de protección de datos, le recomendamos que proteja Cuenta de AWS las credenciales y configure los usuarios 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:

- Utiliza la autenticación multifactor (MFA) en cada cuenta.
- Se utiliza SSL/TLS para comunicarse con AWS los recursos. Exigimos TLS 1.2 y recomendamos TLS 1.3.
- Configure la API y el registro de actividad de los usuarios con AWS CloudTrail. Para obtener información sobre el uso de CloudTrail senderos para capturar AWS actividades, consulte Cómo trabajar con CloudTrail senderos en la Guía del AWS CloudTrail usuario.
- Utilice soluciones de AWS cifrado, junto con todos los controles de seguridad predeterminados Servicios de AWS.

- Utiliza 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 por FIPS 140-3 para acceder a AWS través de una interfaz de línea de comandos o una API, utilice un punto final FIPS. Para obtener más información sobre los puntos de conexión de FIPS disponibles, consulta Estándar de procesamiento de la información federal (FIPS) 140-3.

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 cuando trabajas con Device Farm u otro dispositivo Servicios de AWS mediante la consola AWS CLI, la API o AWS SDKs. Cualquier dato que introduzca en etiquetas o campos de formato libre utilizados para los nombres se pueden 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 finales de Device Farm solo admiten HTTPS () SSL/TLS) requests except where otherwise noted. All content retrieved from or placed in Amazon S3 through upload URLs is encrypted using SSL/TLS firmados. Para obtener más información sobre cómo se inicia sesión en las solicitudes HTTPS AWS, consulte Firmar solicitudes de AWS API en la Referencia AWS general.

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. Cuando venza el periodo de retención, los datos se eliminarán del almacenamiento de copia de seguridad de Device Farm.

Tipo de contenido	Periodo de retención (días)	Periodo de retención de metadatos (días)
Aplicaciones cargadas	30	30
Paquetes de prueba cargados	30	30
Registros	400	400
Grabaciones de video y otros artefactos	400	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 siguientes secciones se proporciona información sobre los AWS pasos necesarios 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 video 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 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 AWS infraestructura no pública asociada a su cuenta (por ejemplo, EC2 instancias de Amazon sin una dirección IP pública) debe utilizar un punto de enlace de Amazon 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 AWS red estén protegidas o seguras, y es su responsabilidad proteger cualquier conexión a Internet que establezcan 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 servicio gestionado, AWS Device Farm está protegido por la seguridad de la red AWS global. Para obtener información sobre los servicios AWS de seguridad y cómo se AWS protege la infraestructura, consulte Seguridad AWS en la nube. Para diseñar su AWS entorno utilizando las mejores prácticas de seguridad de la infraestructura, consulte Protección de infraestructuras en un marco de buena AWS arquitectura basado en el pilar de la seguridad.

Las llamadas a la API AWS publicadas se utilizan para acceder a Device Farm a través de la red. Los clientes deben admitir lo siguiente:

- Seguridad de la capa de transporte (TLS). Exigimos TLS 1.2 y recomendamos TLS 1.3.
- Conjuntos de cifrado con confidencialidad directa total (PFS) 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 principal de IAM. También puedes 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 AWS cuentas que estén explícitamente autorizadas a hacerlo. Device Farm aísla físicamente estos dispositivos de otros dispositivos y los mantiene en una red separada.

En los dispositivos gestionados de forma privada, las pruebas se pueden configurar para usar un punto de conexión de Amazon VPC para proteger las conexiones de entrada y salida de su AWS cuenta.

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 entre sí sin un tercero intermediario, externo a él. AWS

Todo el tráfico a los WebDriver controladores de Selenium debe realizarse a través del punto final HTTPS generado con. `createTestGridUrl`

Usted es responsable de asegurarse de que cada instancia de prueba de Device Farm tiene acceso seguro a los recursos de la prueba. De forma predeterminada, las instancias de prueba del navegador de escritorio de Device Farm tienen acceso al Internet público. Al asociar la instancia a una VPC, esta se comporta como cualquier otra instancia de EC2, con acceso a los recursos determinados por la configuración de la VPC y sus componentes de red asociados. AWS proporciona grupos de seguridad y listas de control de acceso a la red (ACLs) para aumentar la seguridad de su VPC. Los grupos de seguridad controlan el tráfico entrante y saliente de sus recursos, y la red

ACLs controla el tráfico entrante y saliente de sus subredes. Los grupos de seguridad proporcionan suficiente control de acceso para la mayoría de las subredes. Puede usar la red ACLs si desea una capa de seguridad adicional para su VPC. Para obtener directrices generales sobre las prácticas recomendadas de seguridad al utilizar Amazon VPCs, consulte las prácticas recomendadas de seguridad para su VPC en la Guía del usuario de Amazon Virtual Private Cloud.

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 lo acordado con usted. 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 las gemas de Ruby, consulta las prácticas de seguridad en el RubyGems sitio web.
- Para obtener información sobre el paquete de seguridad utilizado por Pipenv y respaldado por la Autoridad de Empaquetado de Python 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 de Maven del Open Web Application Security Project (OWASP), consulte OWASP en el sitio web de OWASP. DependencyCheck

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 tiene conocimiento de un caso en el que otro cliente puede acceder a los datos de un cliente, como los resultados de las pruebas o los archivos escritos en un dispositivo público, se pone en AWS contacto con los clientes afectados de acuerdo con las políticas estándar de alerta e informe de incidentes que se utilizan en todos los servicios. AWS

Registro y supervisión en Device Farm

Este servicio admite AWS CloudTrail, que es un servicio que graba sus AWS llamadas Cuenta de AWS y entrega los archivos de registro a un bucket de Amazon S3. Al usar la información recopilada por CloudTrail, puede determinar qué solicitudes se realizaron correctamente Servicios de AWS, quién realizó la solicitud, cuándo se realizó, etc. Para obtener más información sobre CloudTrail cómo activarlo y encontrar los archivos de registro, consulta la Guía del AWS CloudTrail usuario.

Para obtener información sobre el uso CloudTrail con Device Farm, consulteRegistro 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 constituyen 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 Entornos de pruebas personalizados.

Límites de AWS Device Farm

Temas

Límites de los servicios

- 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 puede incrementarse si se solicita. El equipo de servicio evaluará cada situación caso por caso.
- No existe ningún límite en el número de ejecuciones que puede programar. Tenga en cuenta que solo pueden permanecer en cola durante un máximo de 24 horas.
- Hay un límite estricto de 150 minutos para la duración de una sesión de acceso remoto.
- Hay un límite estricto de 150 minutos para la duración de una ejecución de prueba 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 una ejecución de prueba. El número de dispositivos (o trabajos) en los que se pueden ejecutar pruebas en paralelo en un momento determinado es igual a la simultaneidad en el nivel de cuenta. La simultaneidad predeterminada en el nivel cuenta para el uso medido en Device Farm es 5.
- El límite de simultaneidad medido se puede aumentar, si se solicita, 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.

  Para obtener más información sobre los límites o cuotas de simultaneidad medidos predeterminados en general, consulte la página Cuotas.

- Una ejecución de automatización que no utilice un entorno de pruebas personalizado solo puede contener hasta 250 casos de prueba individuales. De lo contrario, es posible que se omita la ejecución.

Límites de archivos

- El tamaño de archivo máximo de una aplicación que puede cargar es de 4 GB. Tenga en cuenta que actualmente no aceptamos archivos con formato .aab para Android.
- El tamaño máximo del vídeo generado automáticamente por Device Farm durante la prueba es de 1 GB. Cualquier vídeo que supere este tamaño tendrá todo el contenido de vídeo restante truncado. Los clientes pueden seguir utilizando su propia solución de grabación de vídeo, si la tienen, y almacenarla fuera del almacenamiento administrado de Device Farm.
- El tamaño máximo del registro de dispositivos generado automáticamente por Device Farm (logcat en Android o syslog en iOS) durante la ejecución de la prueba es de 1 GB. Cualquier registro que supere este tamaño tendrá todos los registros restantes truncados. Para los registros de más de 1 GB, los clientes pueden almacenar estos registros fuera del almacenamiento administrado de Device Farm.
- El tamaño máximo acumulado de los artefactos de los clientes en modo de entorno personalizado de Device Farm es de 1 GB. Si los artefactos superan este tamaño, ninguno estará disponible.
- Si el tamaño acumulado de todos los artefactos generados durante una prueba supera los 4 GB, es posible que se eliminen algunos artefactos (como el vídeo, los registros de los dispositivos y los artefactos de los clientes).

Límites de la API

- Device Farm sigue un algoritmo de token-bucket para limitar las tasas de llamadas a la API. Por ejemplo, imagine crear un bucket que contenga tokens. Cada token representa una transacción y una llamada a la API consume un token. Los tokens se añaden al bucket a un ritmo fijo (por ejemplo, 10 tokens por segundo) y el bucket tiene una capacidad máxima (por ejemplo, 100 tokens). Cuando llega una solicitud o un paquete, debe reclamar un token del bucket para procesarlo. Si hay suficientes tokens, se permite la solicitud y se eliminan los tokens. Si no hay suficientes tokens, la solicitud se retrasa o se retira, según la implementación.

  En Device Farm, así es como se implementa el algoritmo:

- Ampliación de las solicitudes de la API es la cantidad máxima de solicitudes a las que el servicio puede responder para una API específica en un ID de cuenta de cliente específico. En otras palabras, esta es la capacidad del bucket. Puede llamar a la API tantas veces como tokens queden en el bucket. Cada solicitud consume un token.
- La tasa Transactions-per-second (TPS) es la velocidad mínima a la que se pueden ejecutar las solicitudes de la API. En otras palabras, esta es la velocidad con la que se rellena el bucket con tokens por segundo. Por ejemplo, si una API tiene un número de ampliaciones de diez pero una TPS de uno, podría llamarla diez veces de forma instantánea. Sin embargo, el bucket solo recuperaría los tokens a una velocidad de un token por segundo, por lo que se limitaría a una llamada por segundo, a menos que dejara de llamar a la API para permitir que el bucket se rellene.

Estas son las tarifas de Device Farm APIs:

- En el caso de List and Get APIs, la API Burst solicita la capacidad y la tasa Transactions-per-second (TPS) es`10`. `50`
- Para todas las demás APIs, la API Burst solicita capacidad y la tasa Transactions-per-second (TPS) es`10`. `1`

Límites de punto final de Appium

Los siguientes límites se aplican a todas las sesiones de puntos finales de Appium. Si tiene preguntas o necesita orientación sobre la mejor manera de gestionar los límites, abra un caso de soporte.

- Cada comando de Appium tiene un límite de ejecución de 4 minutos, después del cual se agota el tiempo de espera del comando.
- El terminal acepta tamaños de carga útil de entrada de hasta 20 MB y permite tamaños de carga útil de salida de hasta 20 MB. Cualquier solicitud con un tamaño de entrada o salida mayor que este recibirá un error de. WebDriver `'unsupported operation'`
- Las solicitudes se ejecutan secuencialmente en el dispositivo en el orden en que se reciben. Por ello, recomendamos encarecidamente enviar los comandos de forma secuencial y esperar la respuesta de cada comando antes de enviar uno nuevo. Dicho esto, ciertos comandos del servidor Appium se pueden enviar en paralelo, específicamente:
  - GetStatus

  - Obtenga sesiones
- El punto final no admite el WebDriver BiDi protocolo en este momento.
- El terminal no admite complementos de Appium ni controladores distintos de los controladores XCUITest and UIAutomator2 .
- Se pueden usar un máximo de 3 aplicaciones como aplicaciones auxiliares con una solicitud de creación de sesión de acceso remoto. Dicho esto, no hay límite en cuanto al número de aplicaciones que se pueden instalar durante una sesión mediante la `InstallToRemoteAccessSession`API.

Límites de variables de entorno personalizados

Los siguientes límites se aplican a todas las variables de entorno personalizadas. Si tiene preguntas o necesita orientación sobre la mejor manera de gestionar los límites, abra un caso de soporte.

- Se puede configurar un máximo de 32 variables en un proyecto o ejecución de Device Farm determinado.
- Los nombres de las variables no pueden superar los 256 caracteres.
- Los nombres de las variables están sujetos a las limitaciones impuestas por`bash`. Es decir, deben contener únicamente caracteres alfanuméricos y caracteres de subrayado, y no pueden empezar por un número.
- Los nombres de variables que comiencen por `$DEVICEFARM_` están reservados para el uso interno del servicio.
- Los valores de las variables no pueden superar los 256 caracteres de longitud.
- Las variables de entorno no se pueden usar para configurar la selección de cómputo del host de prueba en el archivo de especificaciones de la prueba.

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

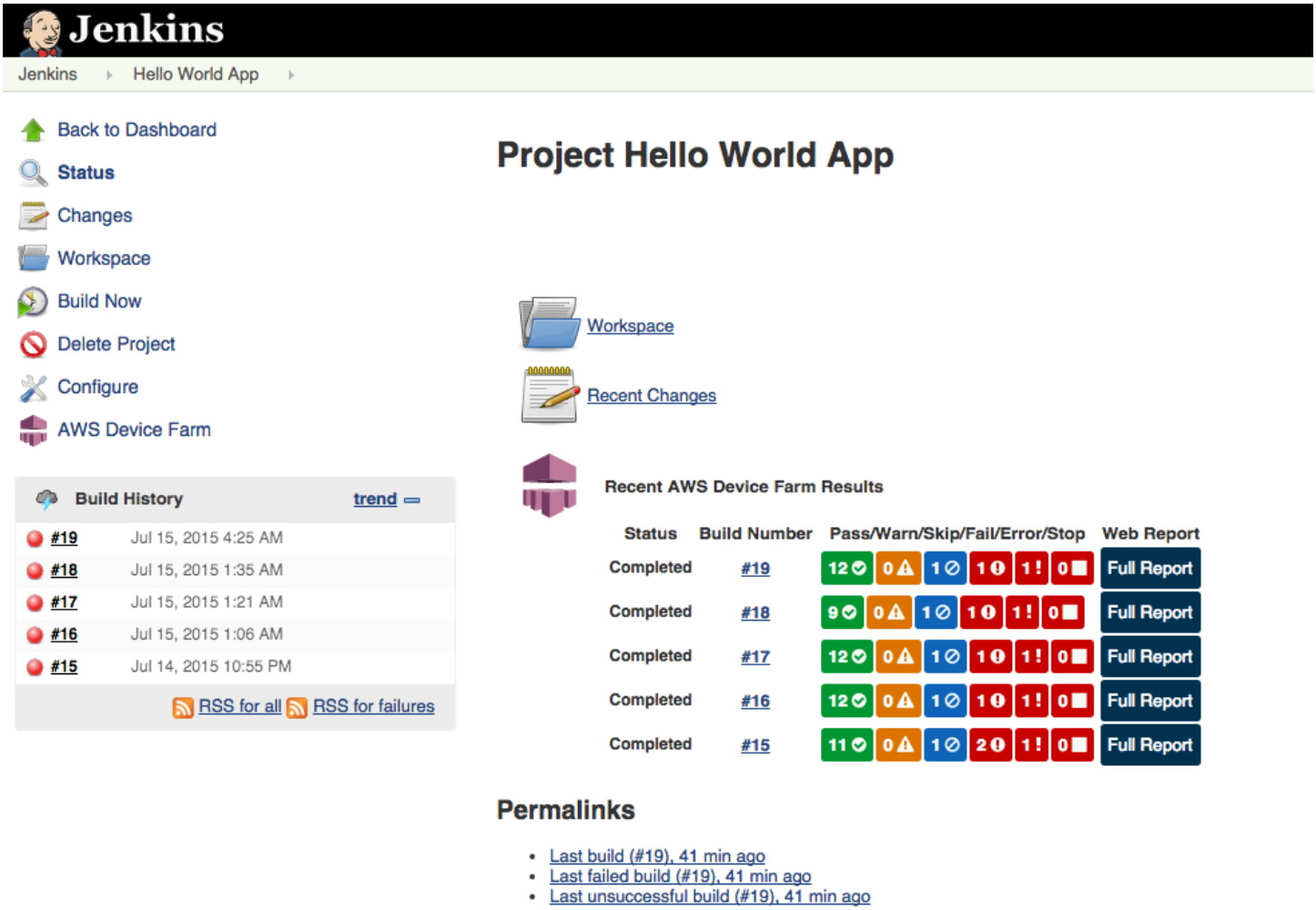
Integración de Device Farm con un servidor Jenkins CI

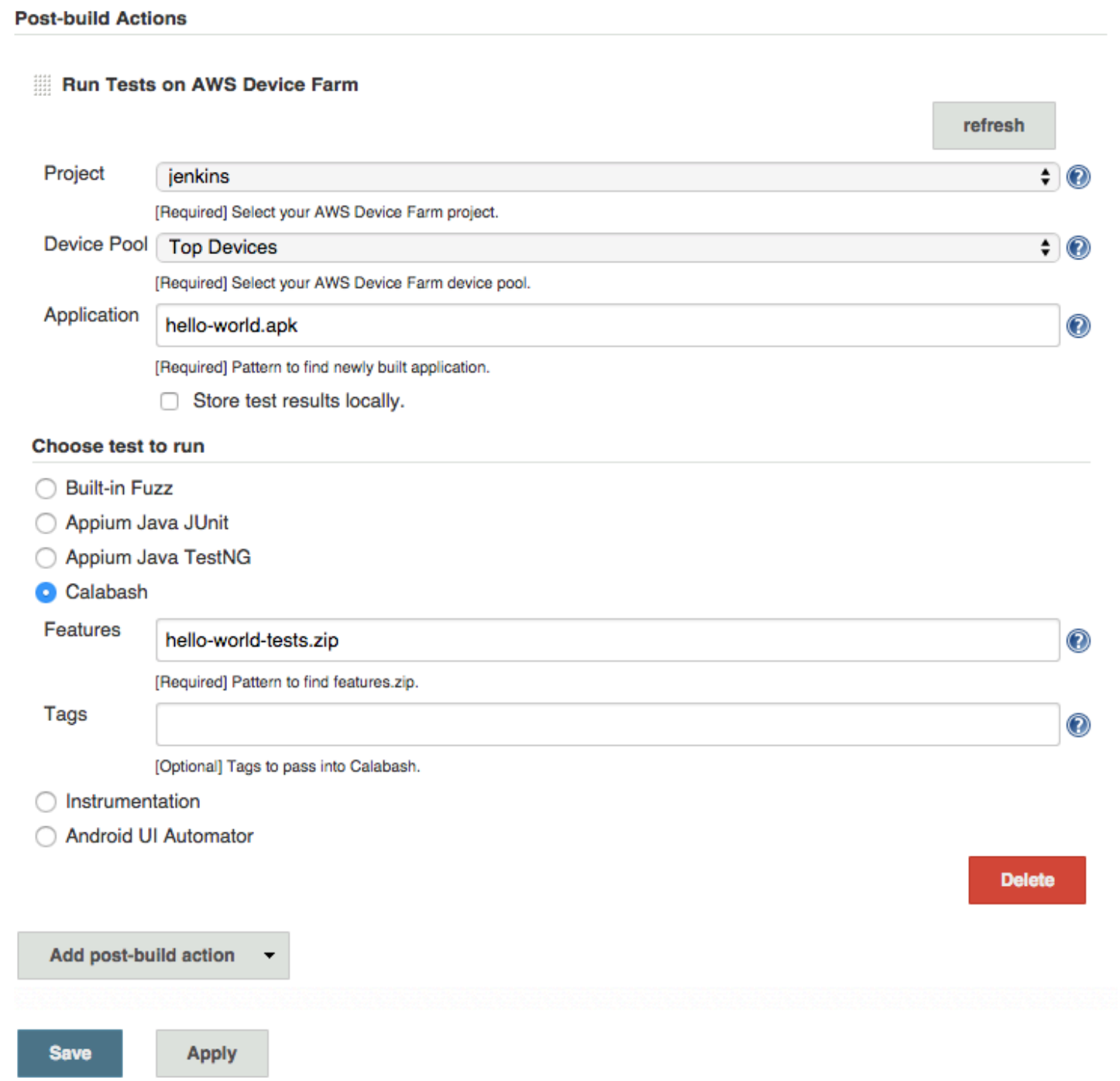
El complemento Jenkins CI proporciona funcionalidad de AWS Device Farm desde su propio servidor Jenkins de integración continua (CI). Para obtener más información, consulte Jenkins (software).

> **Note**
>
> Para descargar el complemento de Jenkins, vaya a GitHuby siga las instrucciones que aparecen enPaso 1: instalación del complemento Jenkins CI para AWS Device Farm.

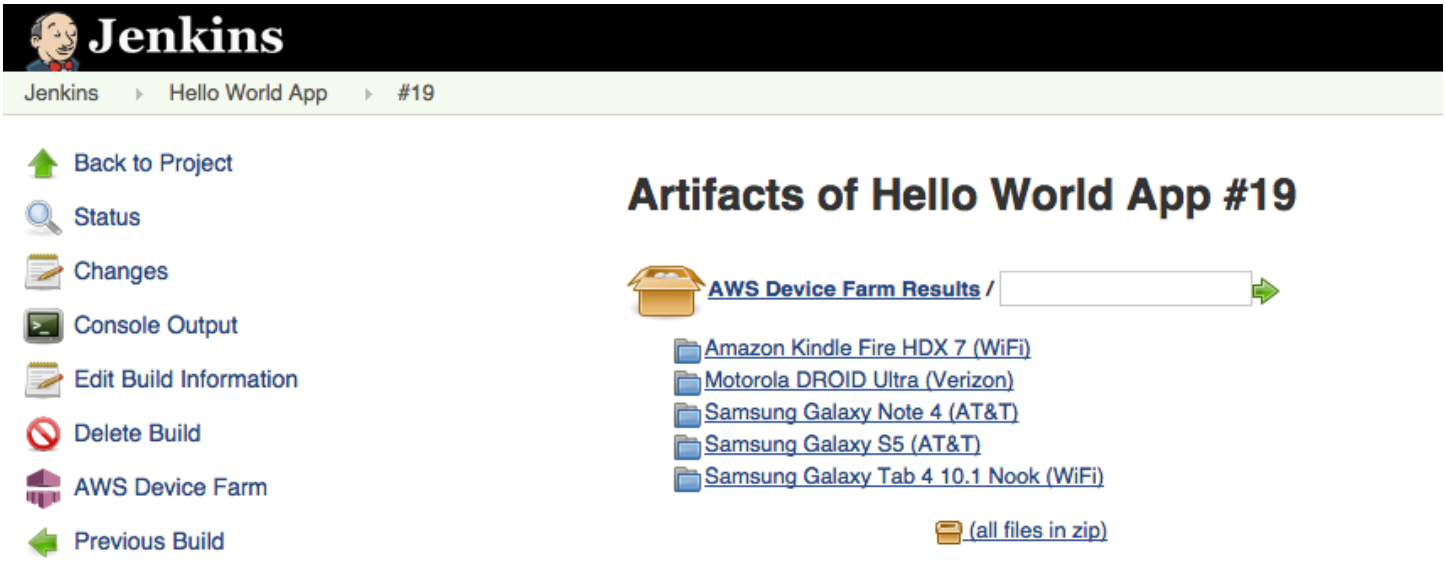
Esta sección contiene una serie de procedimientos para configurar y utilizar el complemento Jenkins CI con AWS Device Farm.

Las siguientes imágenes muestran las características del complemento Jenkins CI.


Jenkins
Jenkins ▸ Hello World App ▸
Back to Dashboard
Status
Changes
Workspace
Build Now
Delete Project
Configure
AWS Device Farm
Build History trend
#19 Jul 15, 2015 4:25 AM
#18 Jul 15, 2015 1:35 AM
#17 Jul 15, 2015 1:21 AM
#16 Jul 15, 2015 1:06 AM
#15 Jul 14, 2015 10:55 PM
RSS for all RSS for failures
Project Hello World App
Workspace
Recent Changes
Recent AWS Device Farm Results
Status Build Number Pass/Warn/Skip/Fail/Error/Stop Web Report
Completed #19 12 0 1 1 1 0 Full Report
Completed #18 9 0 1 1 1 0 Full Report
Completed #17 12 0 1 1 1 0 Full Report
Completed #16 12 0 1 1 1 0 Full Report
Completed #15 11 0 1 2 1 0 Full Report
Permalinks
Last build (#19), 41 min ago
Last failed build (#19), 41 min ago
Last unsuccessful build (#19), 41 min ago



El complemento también puede extraer todos los artefactos de las pruebas (registros, capturas de pantalla, etc.) localmente:



Temas

- Dependencias
- Paso 1: instalación del complemento Jenkins CI para AWS Device Farm
- Paso 2: Crear un AWS Identity and Access Management usuario para el complemento CI de Jenkins para AWS Device Farm
- Paso 3: configuración del complemento Jenkins CI por primera vez en AWS Device Farm
- Paso 4: uso del complemento en un trabajo de Jenkins

Dependencias

El complemento Jenkins CI requiere la AWS versión 1.10.5 o posterior del SDK para dispositivos móviles. Para obtener más información y para instalar el SDK, consulte AWS Mobile SDK.

Paso 1: instalación del complemento Jenkins CI para AWS Device Farm

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 la opción 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 `hpi` archivo 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 AWS Identity and Access Management usuario para el complemento CI de Jenkins para AWS Device Farm

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

Para crear un nuevo usuario de IAM, consulte Crear un usuario de IAM (Consola de administración de AWS). 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 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

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:

   JSON

```
{
    "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: configuración del complemento Jenkins CI por primera vez en AWS Device Farm

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 Creación de un usuario de IAM para el complemento Jenkins CI y pegue el ID de clave de acceso y la clave de acceso secreta en sus respectivas casillas.
6. Seleccione Save.

Paso 4: uso del 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.

Integración de Device Farm con un sistema de compilación Gradle

El complemento Gradle de Device Farm 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 de Gradle, ve a GitHuby sigue las instrucciones que aparecen enCreación 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

Dependencias

Tiempo de ejecución

- El complemento Device Farm para Gradle requiere el SDK AWS móvil 1.10.15 o una versión 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

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. Ve al repositorio GitHuby clónalo.
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: Generación de un usuario de IAM en el complemento Gradle de Device Farm

Paso 3: generación de un usuario de IAM en el complemento Gradle de Device Farm.

AWS Identity and Access Management (IAM) le ayuda a administrar los permisos y las políticas para trabajar con AWS los recursos. 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 utilice su cuenta AWS root para acceder 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 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.

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. Seleccione 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 Seleccionar.
13. Especifique un nombre para la política, como **`AWSDeviceFarmGradlePolicy`**.
14. En Documento de política, pegue la siguiente política.

    JSON

```
{
    "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 Consola de administración de AWS() o Configuración.

Paso 4: Configurar los tipos de prueba

De forma predeterminada, el complemento Gradle de AWS Device Farm ejecuta la prueba 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 Marcos de pruebas y pruebas integradas 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 y JUnit 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 Ejecución de la prueba de difusión integrada de Device Farm (Android e iOS).

Instrumentación

Device Farm ofrece soporte para instrumentación (EspressoJUnit, Robotium o cualquier prueba basada en instrumentación) para Android. Para obtener más información, consulte 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

    }
```

Historial de documentos de AWS Device Farm

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
Soporte para terminales de Appium	Device Farm ahora ofrece un terminal Appium totalmente gestionado para realizar pruebas de dispositivos de forma remota, lo que permite desarrollar y depurar rápidamente las pruebas. Esto complementa el método de ejecución del lado del servidor existente, en el que las pruebas se cargan y ejecutan directamente en Device Farm. Si bien la ejecución en el lado del servidor es ideal para los procesos y CI/CD las pruebas a gran escala, el nuevo terminal local de Appium permite una iteración y un desarrollo más rápidos de las pruebas en dispositivos reales.	17 de noviembre de 2025
Mejoras en el host de pruebas de iOS	Device Farm ahora admite una experiencia actualiza da para el entorno de pruebas de iOS, lo que permite la coherencia en las configuraciones entre las pruebas de Android e iOS. Consulte Hosts para entornos de prueba personalizados para obtener más información. Además, se ha eliminado la información relacionada con los anfitriones de pruebas de Android retirados. Se recomienda a los usuarios de Android que utilicen los hosts de prueba de Amazon Linux 2.	31 de octubre de 2025
AL2 apoyo	Device Farm ahora es compatible con el entorno de AL2 pruebas de Android. Obtener más información sobre AL2.	6 de noviembre de 2023
Migración de entornos de prueba estándar a entornos	Se ha actualizado la guía de migración para documenta r la obsolescencia de las pruebas de modo estándar en diciembre de 2023.	3 de septiembre de 2023

Cambio	Descripción	Fecha de modificación
de prueba personalizados		
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 el etiquetado de AWS recursos.	Para que AWS los servicios de protección sean más fáciles y completos, se ha creado una nueva sección sobre seguridad. 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.	27 de marzo de 2020
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

Cambio	Descripción	Fecha de modificación
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 para las pruebas con XCTest	Para las ejecuciones de prueba que utilizan el XCTest marco, Device Farm ahora requiere un paquete de aplicaciones creado para las 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 Marcos de pruebas y pruebas integradas en AWS Device Farm.	10 de enero de 2019
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 Marcos de pruebas y pruebas integradas en AWS Device Farm.	2 de octubre de 2018

Cambio	Descripción	Fecha de modificación
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 video, de modo que puede obtener comentarios instantáneos sobre las pruebas que se ejecutan en un entorno de pruebas personalizado. Obtener más información sobre Personalización del entorno de pruebas personalizado en AWS Device Farm..	16 de agosto de 2018
Support para usar Device Farm como proveedor AWS CodePipeline de pruebas	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 Integración de AWS Device Farm en una fase CodePipeline de prueba.	19 de julio de 2018
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 configuraciones de estos dispositivos, crear puntos de conexión de VPC de Amazon para probar aplicaciones privadas y crear las sesiones de depuración remota. Obtener más información sobre 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

Cambio	Descripción	Fecha de modificación
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 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 Acceso remoto.	19 de abril de 2016
Autoservicio de ranuras de dispositi vos	Ahora puede comprar ranuras para dispositivos mediante la Consola de administración de AWS AWS Command Line Interface, la o la API. Obtenga más información sobre cómo Adquisición de una ranura de dispositivos en Device Farm.	22 de marzo de 2016
Cómo detener ejecuciones de prueba	Ahora puede detener las ejecuciones de prueba mediante la Consola de administración de AWS AWS Command Line Interface, la o la API. Obtenga más información sobre cómo Detención de una ejecución en AWS Device Farm.	22 de marzo de 2016

Cambio	Descripción	Fecha de modificación
Nuevos tipos de pruebas de XCTest interfaz de usuario	Ahora puedes ejecutar pruebas personalizadas de XCTest interfaz de usuario en aplicaciones iOS. Obtenga más información sobre el tipo de prueba de Integración de la XCTest interfaz de usuario para iOS con Device Farm.	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 Marcos de pruebas y pruebas integradas 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 Pruebas de aplicaciones web en AWS Device Farm.	19 de noviembre de 2015
Complemento Gradle de 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 sobre las pruebas de dispositi vos iOS y la ejecución de pruebas de iOS (incluidas XCTest) enMarcos de pruebas y pruebas integradas en AWS Device Farm.	4 de agosto de 2015
Versión pública inicial	Esta es la versión pública inicial de la Guía para desarroll adores de AWS Device Farm.	13 de julio de 2015

AWSGlosario de

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.